WO9728505

Publication Title:

IMPROVED METHOD AND APPARATUS FOR DYNAMICALLY SHIFTING BETWEEN ROUTING AND SWITCHING PACKETS IN A TRANSMISSION NETWORK

Abstract:

Abstract of WO 9728505

(A1) A method and apparatus for dynamically shifting between switching and routing packets efficiently to provide high throughput. The present invention provides a method for transmitting packets betwee an upstream node and a downstream node in a network that utilizes flow classification and labelling to redirect flows. The method includes the steps of establishing default virtual channels between the upstream node and the downstream node (1602), receiving packets at the downstream node, performing flow classification at the downstream node on the packet to determine whether the packet belongs to a specified flow that should be redirected in the upstream node (170), selecting a free label at the downstream node (1668) and informing the upstream node that future packets belonging to the specified flow should be sent with the selected free label attached (1676). Other embodiments of the invention include a basic switching unit (1, 3), a switch gateway unit (21), and a switching agent for use in a system for transmitting packets in a network.

Courtesy of http://v3.espacenet.com

This Patent PDF Generated by Patent Fetcher(R), a service of Stroke of Color, Inc.

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 13/14, 13/376, H04L 12/28, 12/407, 12/66, 12/56 — A1

(11) International Publication Number: WO 97/28505

(43) International Publication Date: 7 August 1997 (07.08.97)

(21) International Application Number: PCT/US97/01595

(22) International Filing Date: 30 January 1997 (30.01.97)

(30) Priority Data:

08/597,520	31 January 1996 (31.01.96)	US
60/024,272	22 November 1996 (22.11.96)	US

(71) Applicant: IPSILON NETWORKS, INC. [US/US]; 232 Java Drive, Sunnyvale, CA 94089 (US).

(72) Inventors: LYON, Thomas; 1400 Edgewood Drive, Palo Alto, CA 94301 (US). NEWMAN, Peter; Apartment 124, 750 North Shoreline Boulevard, Mountain View, CA 94043 (US). MINSHALL, Greg; 553 Tyndall Lane, Los Altos, CA 94022 (US). HINDEN, Robert; 3217 Murray Way, Palo Alto, CA 94303 (US). LIAW, Fong, Ching; 775 Ajax Drive, Sunnyvale, CA 94086 (US). HOFFMAN, Eric; 740 - 6th Avenue, Redwood City, CA 94063 (US). HUSTON, Lawrence, B.; Apartment 9, 237 Arriba Drive, Mountain View, CA 94086 (US). ROBERSON, William, A.; 892 Lockwood Lane, Scotts Valley, CA 95066 (US).

(74) Agents: KIM, Rosa, S. et al.; Townsend and Townsend and Crew L.L.P., 8th floor, Two Embarcadero Center, San Francisco, CA 94111-3834 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

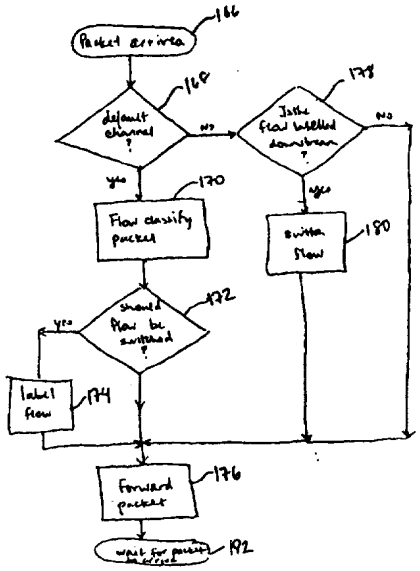
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: IMPROVED METHOD AND APPARATUS FOR DYNAMICALLY SHIFTING BETWEEN ROUTING AND SWITCHING PACKETS IN A TRANSMISSION NETWORK

(57) Abstract

A method and apparatus for dynamically shifting between switching and routing packets efficiently to provide high throughput. The present invention provides a method for transmitting packets betwee an upstream node and a downstream node in a network that utilizes flow classification and labelling to redirect flows. The method includes the steps of establishing default virtual channels between the upstream node and the downstream node (1602), receiving packets at the downstream node, performing flow classification at the downstream node on the packet to determine whether the packet belongs to a specified flow that should be redirected in the upstream node (170), selecting a free label at the downstream node (1668) and informing the upstream node that future packets belonging to the specified flow should be sent with the selected free label attached (1676). Other embodiments of the invention include a basic switching unit (1, 3), a switch gateway unit (21), and a switching agent for use in a system for transmitting packets in a network.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	**GB**	United Kingdom	**MW**	Malawi
AT	Austria	**GE**	Georgia	**MX**	Mexico
AU	Australia	**GN**	Guinea	**NE**	Niger
BB	Barbados	**GR**	Greece	**NL**	Netherlands
BE	Belgium	**HU**	Hungary	**NO**	Norway
BF	Burkina Faso	**IE**	Ireland	**NZ**	New Zealand
BG	Bulgaria	**IT**	Italy	**PL**	Poland
BJ	Benin	**JP**	Japan	**PT**	Portugal
BR	Brazil	**KE**	Kenya	**RO**	Romania
BY	Belarus	**KG**	Kyrgystan	**RU**	Russian Federation
CA	Canada	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CF	Central African Republic			**SE**	Sweden
CG	Congo	**KR**	Republic of Korea	**SG**	Singapore
CH	Switzerland	**KZ**	Kazakhstan	**SI**	Slovenia
CI	Côte d'Ivoire	**LI**	Liechtenstein	**SK**	Slovakia
CM	Cameroon	**LK**	Sri Lanka	**SN**	Senegal
CN	China	**LR**	Liberia	**SZ**	Swaziland
CS	Czechoslovakia	**LT**	Lithuania	**TD**	Chad
CZ	Czech Republic	**LU**	Luxembourg	**TG**	Togo
DE	Germany	**LV**	Latvia	**TJ**	Tajikistan
DK	Denmark	**MC**	Monaco	**TT**	Trinidad and Tobago
EE	Estonia	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**UG**	Uganda
FI	Finland	**ML**	Mali	**US**	United States of America
FR	France	**MN**	Mongolia	**UZ**	Uzbekistan
GA	Gabon	**MR**	Mauritania	**VN**	Viet Nam

IMPROVED METHOD AND APPARATUS FOR DYNAMICALLY SHIFTING BETWEEN ROUTING AND SWITCHING PACKETS IN A TRANSMISSION NETWORK

BACKGROUND OF THE INVENTION

The present invention relates to the field of network communications. More particularly, in one embodiment the present invention provides a method and apparatus for dynamically shifting between switching and routing packets efficiently to provide high packet throughput while maintaining complete Internet Protocol (IP) routing functionality. The present invention combines high speed, capacity, multiservice traffic capability, with simplicity, scaleability, and robustness.

Due to the current popularity and continual growth of the Internet, which utilizes IP, IP has evolved into the dominant network-layer protocol in use today. IP specifies protocol data unit (PDU) format and station-router and router-router interaction. IP provides a connectionless data transfer service to IP users in stations attached to networks of the Internet. The connectionless model on which IP is based provides a robust and flexible basis on which to construct an integrated services network. All major operating systems include an implementation of IP, enabling IP and its companion transport-layer (Layer 4 of the OSI reference model) protocol, the Transmission Control Protocol (TCP), to be used universally across virtually all hardware platforms. One of the major advantages of IP is its tremendous scaleability, operating successfully in networks with only a few users to enterprise-size networks, including the global Internet.

With the rapid growth of the Internet, conventional IP routers are becoming inadequate in their ability to handle the traffic on the Internet. With today's faster workstations, client-server computing, and higher bandwidth requirement applications, networks are increasingly encountering traffic congestion problems. Typical problems include for example highly variable network response

times, higher network failure rates, and the inability to support delay-sensitive applications.

Local area network (LAN) switches offer a quick, relatively inexpensive way to relieve congestion on shared-media LAN segments. Switching technology is emerging as a more effective means of managing traffic and allocating bandwidth within a LAN than shared-media hubs or simple bridges. LAN switches operate as datalink layer (Layer 2 of the OSI reference model) packet-forwarding hardware engines, dealing with media access control (MAC) addresses and performing simple table look-up functions. Switch-based networks are able to offer greater throughput, but they continue to suffer from problems such as broadcast flooding and poor security. Routers, which operate at the network-layer (Layer 3 of the OSI reference model), are still required to solve these types of problems. However, fast switching technology is overwhelming the capabilities of current routers, creating router bottlenecks. The traditional IP packet-forwarding device on which the Internet is based, the IP router, is showing signs of inadequacy. Routers are expensive, complex, and of limited throughput, as compared to emerging switching technology. To support the increased traffic demand of large enterprise-wide networks and the Internet, IP routers need to operate faster and cost less.

Additionally, quality of service (QOS) selection is needed in order to support the increasing demand for real-time and multimedia applications, including for example conferencing. Currently TCP/IP does not support QOS selection. However, as advanced functionalities required by more types of traffic are enabled in IP, traditional IP routers will not suffice as packet-forwarding devices.

Asynchronous transfer mode (ATM) is a high-speed, scaleable, multiservice technology touted as the cornerstone of tomorrow's router-less networks. ATM is a highly efficient packet-forwarding technology with very high throughput, scaleability, and support for multiple types of traffic including voice and video as well as data. However, ATM is a networking technology so different from current networking architectures such as IP that there is no clear migration path to it. ATM has difficulty in effectively supporting existing LAN traffic due to its connection-oriented architecture, which creates the need for an additional set of very complex, untested multi-layer protocols. Problems with these protocols are

evidenced by unacceptably long switched virtual circuit (SVC) connection setup times. Additionally, enabling TCP/IP users to send and receive ATM traffic using SVCs requires adopting even more new, unproven, and extremely complex protocols. These protocols do not enable applications running on TCP/IP protocols to take advantage of the QOS features of ATM, thereby imposing a tremendous amount of overhead for network managers without enabling one of the key benefits of ATM. Also, many of these protocols duplicate the functionality of the well-established TCP/IP protocol suite, and the need to learn these complex protocols increases the costs of ownership of ATM devices for network managers who must troubleshoot problems in the network. The difficulties of moving to ATM are especially pronounced in light of the time-tested and debugged IP being solidly entrenched with its huge and growing installed user base as evidenced by the popularity of the Internet.

In response to the inadequacies of current solutions to the problems, vendors have developed a host of new distributed routing networking architectures. However, these architectures are often complex, confusing, and duplicative of functionalities provided by IP. These architectures also result in increasingly complex problems for network managers. For example, duplication of functionality leads to increased strain on the network management function and can make isolation of network problems very difficult. It is seen that a system for high speed routing is needed to avoid bottlenecks and increased network management complexity. Further, provision of a networking architecture having compatibility with IP without unnecessary duplication is needed.

SUMMARY OF THE INVENTION

The present invention relates to the field of network communications, and in particular provides a method and apparatus for dynamically shifting between switching and routing packets efficiently to provide high packet throughput to solve the problems discussed above.

According to an embodiment, the present invention provides a method for transmitting packets between an upstream node and a downstream node in a network, the downstream node being downstream from the upstream node.

The method includes the steps of establishing default virtual channels between the upstream node and the downstream node, receiving a packet at the downstream node, and performing a flow classification at the downstream node on the packet to determine whether the packet belongs to a specified flow that should be redirected in the upstream node. The method also includes selecting a free label at the downstream node, and informing the upstream node that future packets belonging to the specified flow should be sent with the selected free label attached.

In another embodiment, the present invention provides a method for switching a flow at a first node, the first node having a downstream link to a second node and an upstream link to a third node. The method includes the steps of performing a flow classification at the first node on a first packet to determine whether the first packet belongs to a specified flow that should be redirected in the third node, selecting a first free label at the first node, informing the third node that future packets belonging to the specified flow should be sent with the selected first free label attached. The method also includes performing a flow classification at the second node on a second packet to determine whether the second packet belongs to the specified flow that should be redirected in the third node, selecting a second free label at the second node, and informing the first node that future packets belonging to the specified flow should be sent with the selected second free label attached. The method operates such that the specified flow from the upstream link may be switched in layer 2 by the first node to the downstream link.

According to another embodiment, the present invention provides a basic switching unit in a system for transmitting packets in a network. The basic switching unit includes switching hardware, and a controller coupled to the switching hardware. The controller, which includes a processor and memory, controls the switching hardware. The basic switching unit further includes software, fixed on tangible media, that enables the basic switching unit to dynamically shift between Layer 3 IP packet routing and Layer 2 switching to optimize packet traffic throughput.

In accordance with yet another embodiment, the present invention provides a switch gateway unit in a system for transmitting packets in a network. The system includes a basic switching unit coupled to the switch gateway unit via a

communication link. The switch gateway unit includes a gateway controller, and software. The gateway controller includes a processor, memory, and multiple NICs. The software, fixed on tangible media, enables the switch gateway unit to redirect a flow of packets to a basic switching unit to enable dynamic shifting between packet routing and switching to optimize packet traffic throughput.

In accordance with still another embodiment, the present invention provides a switching agent in a system for transmitting packets in a network. The system includes a basic switching unit coupled to the switching agent via a communication link, where the basic switching unit includes a controller and a switching engine. The switching agent includes a processor, memory, and multiple of NICs, a specific one of these NICs providing the communication link and at least one of these NICs connectable to at least one node in the network. The switching agent also includes computer-readable program code, fixed on a tangible computer-readable media of the memory. The computer-readable program code enables the controller of the basic switching unit to classify a flow and to redirect that flow of packets from a first node to a second node in the network, and also enables the controller of the basic switching unit to instruct the switching agent to perform packet forwarding of that flow from the first node to the second node via the switching engine. Accordingly, packet forwarding is offloaded from the controller of the basic switching unit.

These and other embodiments of the present invention, as well as its advantages and features, are described in more detail in conjunction with the text below and the attached figures.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1a is a simplified diagram of a basic switching unit of the system according to an embodiment of the invention; Fig. 1b is a simplified diagram of a switch gateway unit of the system according to another embodiment of the invention;

Fig. 1c is a simplified diagram of a switching agent of the system according to still another embodiment of the invention;

Figs. 2a-2c are simplified diagrams of exemplary network configurations according embodiments of the present invention;

Fig. 3 is a general system block diagram of an exemplary computer system used according to embodiments of the invention;

Fig. 4 is a general block diagram of an exemplary ATM switch according to an embodiment of the invention;

Fig. 5a is a simplified diagrams generally illustrating the initialization procedure in each system node according to an embodiment of the present invention;

Fig. 5b is a simplified diagram that generally illustrates the operation of a system node;

Fig. 5c is a simplified diagram generally illustrating the procedure at a switching agent when a packet arrives on one of its interfaces after initialization;

Fig. 5d is a simplified diagram generally illustrating the procedure at a switch controller (to which at least one switching agent may be attached via a communication link, for example, using the switching engine of the switch controller) when a packet arrives from a switching agent on one of its interfaces on a default channel, after initialization;

Fig. 6a is a diagram generally illustrating the steps involved in labelling a flow in a system node;

Fig. 6b is a diagram generally illustrating the steps involved in switching a flow in a basic switching unit;

Fig. 6c is a diagram generally illustrating the steps involved in forwarding a packet in a system node (or switching node); Fig. 6d is a diagram generally illustrating the steps the performed in the switch controller in labelling a flow for packets received from a source switching agent in three scenarios;

Fig. 6e is a diagram generally illustrating the steps performed in the switch controller in labelling a flow for packets, which are received from an attached switching node and intended for an interface on an attached switching agent;

Figs. 7a-7b illustrate the formats of flow identifiers for Flow Type 1 and Flow Type 2;

Fig. 8a illustrates the structure of a generic IFMP adjacency protocol message, according to an embodiment of the present invention;

Fig. 8b illustrates a generic IP packet (in its current version IPv4) with a variable length Data field into which an IFMP message may be encapsulated;

Fig. 8c is a simplified diagram illustrating the operation of a system node upon receiving a packet with an incoming IFMP adjacency protocol message;

Fig. 8d is a state diagram illustrating the operation of a sender system node when the incoming IFMP adjacency protocol message is not an RSTACK message;

Fig. 9a illustrates the structure of a generic IFMP redirection protocol message, according to an embodiment of the present invention;

Fig. 9b is a general diagram describing the operation of a system node upon receiving an IFMP redirection protocol message;

Figs. 9c-9g illustrate the structures for a REDIRECT message element, RECLAIM message element, RECLAIM ACK message element, LABEL RANGE message element, and ERROR message element in the Message Body 394 of the respective IFMP redirection protocol messages;

Fig. 10a illustrates the format of a Label field on an ATM data link, according to an embodiment of the present invention;

Fig. 10b-10e respectively illustrate default, Flow Type 0, Flow Type 1, and Flow Type 2 encapsulated IP packets, according to embodiments of the present invention;

Fig. 11a illustrates the format of an encapsulated GSMP packet;

Fig. 11b illustrates the format of a GSMP adjacency protocol message;

Fig. 11c is a simplified diagram illustrating the operation of a sender entity upon receiving a packet with an incoming GSMP adjacency protocol message;

Fig. 11d is a state diagram illustrating the operation of a sender entity when the incoming IFMP adjacency protocol message is not an RSTACK message;

Fig. 12 illustrates the format of a generic GSMP Connection Management message;

Figs. 13a-13e are simplified diagrams illustrating the operation of a receiver entity upon receiving GSMP Connection Management Add Branch, Delete Branch, Delete Tree, Verify Tree, and Delete All messages respectively;

Fig. 13f illustrates the format of a GSMP Connection Management Move Root message;

Fig. 13g is a simplified diagram illustrating the operation of a sender entity upon receiving a packet with an incoming GSMP Connection Management Move Root message;

Fig. 13h illustrates the format of a GSMP Connection Management Move Branch message;

Fig. 13i is a simplified diagram illustrating the operation of a sender entity upon receiving a packet with an incoming GSMP Connection Management Move Branch message;

Fig. 14 illustrates the format of a GSMP Port Management message;

Fig. 15a illustrates an encapsulated IFMP-C packet 1000;

Fig. 15b illustrates the generic structure of a typical IFMP-C message 1012 that may be contained in IFMP-C Message field 1006 of the encapsulated IFMP-C packet 1000 in Fig. 15a;

Fig. 16a illustrates the generic structure of an IFMP-C adjacency protocol message 1040 that may be contained in IFMP-C Message field 1006 of the encapsulated IFMP-C packet 1000 in Fig. 15a;

Fig. 16b is a state diagram illustrating the operation of a sender entity (either an IFMP-C controller or an IFMP-agent) in the three possible states of the IFMP-C adjacency protocol;

Figs. 17a and 17b illustrate the structure of IFMP-C Interface List request and response messages, respectively;

Figs. 17c and 17d illustrate the structure of IFMP-C Interface Query request and response messages, respectively;

Fig. 17e illustrates the structure of an IFMP-C Interface Configuration request message 1170;

Fig. 18a illustrates the message format 1200 of IFMP-C Add Branch request messages and IFMP-C Delete Branch request messages;

Fig. 18b illustrates the Data Transformation field 1240 for a "Truncate packet" transformation type in an IFMP-C Add Branch request message and IFMP-C Delete Branch request message of Fig. 18a;

Fig. 18c illustrates the message format 1250 of an IFMP-C Add Branch response message and an IFMP-C Delete Branch response message;

Fig. 18d illustrates the structure of an IFMP-C Delete Tree request message 1260;

Fig. 18e illustrates the structure of an IFMP-C Move Branch request message 1300;

Fig. 19a illustrates the structure of an IFMP-C Get Tree Statistics request message 1400;

Fig. 19b illustrates the Tree Data field structure 1406, which Tree Data fields use;

Figs. 20a and 20b illustrate the structure of IFMP-C Read Branch request message 1420 and IFMP-C Read Branch response messages 1430, respectively;

Fig. 21a illustrates the structure of IFMP-C Node Information request message 1440;

Figs. 21b and 21c illustrate the structure of IFMP-C Interface Statistics request message 1460 and IFMP-C Interface Statistics response message 1470, respectively;

Fig. 21d illustrates the structure of the Interface Statistics field 1480 in the IFMP-C Interface Statistics response message 1470 of Fig. 21c;

Fig. 21e illustrates the structure of the General Statistics field 1494 within the Interface Statistics field 1480 of the IFMP-C Interface Statistics response message 1470 of Fig. 21c;

Fig. 21f illustrates the structure of the Specific Statistics field 1530 (for an ATM interface) within the Interface Statistics field 1480 of the IFMP-C Interface Statistics response message 1470 of Fig. 21c; and

Fig. 21g illustrates the structure of the Specific Statistics field 1540 (for an Ethernet interface) within the Interface Statistics field 1480 of the IFMP-C Interface Statistics response message 1470 of Fig. 21c.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

CONTENTS

I. General

II. System Hardware

A. Controller Hardware

B. Switching Hardware

C. Exemplary Hardware

III. System Software Functionality

A. IFMP and Transmission of Flow Labelled Packets

B. GSMP

C. IFMP-C

IV. Conclusion

I. General

An improved method and apparatus for transmitting packets in a network are disclosed herein. The method and apparatus will find particular utility and is illustrated herein as it is applied in the high throughput transmission of IP packets capable of carrying voice, video, and data signals over a local area network (LAN), metropolitan area networks (MAN), wide area network (WAN), Internet, or the like, but the invention is not so limited. The invention will find use in a wide variety of applications where it is desired to transmit packets over a network.

The system described herein is a dynamic switching and routing system. The system is described generally as a "switching system," however it should be recognized that the system dynamically provides both switching functionality at the datalink layer 2 as well as routing and packet forwarding functionality at the network layer 3. Additionally, the "basic switching unit" of the system also dynamically provides both layer 2 switching functionality as well as layer 3 routing and packet forwarding functionality. A "switch gateway unit" of the system serves as an access device to enable connection of existing LAN and backbone environments to a network of basic switching units. Similarly to the switch gateway unit, a "switching agent" also serves as an access device to enable connection of existing LAN and backbone environments to at least one basic

switching unit. Both the switch gateway unit and the basic switching unit have independent flow redirect management capability, run routing protocols, and make routing decisions independently in the absence of any flow redirects, as discussed further below. A switch gateway unit and a basic switching unit are therefore peers. In contrast, a switching agent, not having independent flow redirect management capability, forwards packets based on instructions from the basic switching unit acting as master to the slave switching agent. Operating under such instructions from the basic switching unit, the switching agent can forward packets received from the basic switching unit such that a large portion of the packets forwarded by the basic switching unit can now be forwarded by the agent to existing LAN and backbone environments on the agent's interfaces. These environments may include Ethernet, FastEthernet, FDDI, Gigabit Ethernet, or other types of LANs. Since this packet forwarding is performed by the switching agent based on packet forwarding instructions, the basic switching unit is allowed to have more time to perform other tasks such as running routing protocols, as well as reducing the latency for forwarded packets. Performance of packet forwarding by a switching agent reduces the load on the switch controller of the basic switching unit. Accordingly, in some situations where independent flow redirect management capability is not required at a certain node or where the capabilities of the basic switching unit can be better utilized, a switching agent may be suitable for use. A switching agent also may be used as a lower cost substitute for a switch gateway unit. The system is compatible with the Internet Protocol (IP) in its current version (IPv4) as well as with future versions (e.g., IPv6). The system provides dynamic shifting between switching and routing of packets over the network to provide optimal high-speed packet throughput while avoiding router bottlenecks.

As shown in Fig. 1a, a basic switching unit 1 of the switching system, according to an embodiment of the present invention, includes a switching engine 3, a switch controller 5, and system software 7 installed on switch controller 5. In particular, switching engine 3 utilizes conventional and currently available asynchronous transfer mode (ATM) switching hardware. Of course, other switching technologies such as for example fast packet switching, frame relay, Gigabit Ethernet technology or others may be used to provide the switching engine

3 of the present invention, depending on the application. In the present embodiment, switching engine 3 is an ATM switch. Any of the software normally associated with the ATM switch that is above the ATM Adaptation Layer type 5 (AAL-5) is completely removed. Thus, the signalling, any existing routing protocol, and any LAN emulation server or address resolution servers, etc. are removed. Switch controller 5 is a computer having an ATM network adapter or network interface card (NIC) 9 connected to switching engine 3 via an ATM link 11. System software 7 is installed in basic switching unit 1, more particularly in the computer serving as switch controller 5.

Switching engine 3 of basic switching unit 1 has multiple physical ports 13_i capable of being connected to a variety of devices, including for example data terminal equipment (DTE), data communication equipment (DCE), servers, switches, gateways, etc. Each of the physical ports 13_i may be connected via an ATM link to a device equipped with an ATM adapter or NIC, or to a port of another basic switching unit, or to a port of a switch gateway unit, or to a port of a switching agent. The ATM switching hardware providing the switching engine 3 of the basic switching unit operates at the datalink layer (Layer 2 of the OSI reference model).

Switching engine 3 serves to perform high-speed switching functions when required by the basic switching unit, as determined by the system software 7. The switching capability of the switching system is limited only by the hardware used in the switching engine 3. Accordingly, the present embodiment of the invention is able to take advantage of the high-speed, high capacity, high bandwidth capabilities of ATM technology. Of course, other switching technologies such as for example fast packet switching, frame relay, Gigabit Ethernet technology, or others may be used to provide the switching engine 3 of the present invention, depending on the application.

In an embodiment of the present invention, the switch controller 5 is a computer connected to the ATM switch hardware 3 via an ATM link 9, and the system software is installed on the computer. In addition to performing standard connectionless IP routing functions at Layer 3, switch controller 5 also makes flow classification decisions for packets on a local basis.

As shown in Fig. 1b, a switch gateway unit 21 of the switching system, according to another embodiment of the present invention, includes a gateway switch controller 23, and system software 25 installed on gateway switch controller 23. Gateway switch controller 23 includes multiple network adaptors or NICs 27, and an ATM NIC 29. Similar to switch controller 5 of the basic switching unit 1, gateway switch controller 23 also is a computer equipped with an ATM NIC 29 having system software 25 installed on the computer. As discussed above, switch gateway unit 21 serves as an access device to enable connection of existing LAN and backbone environments to a network of basic switching units. Accordingly, NICs 27 may be of different types, such as for example 10BaseT Ethernet NICs, 100BaseT Ethernet NICs, Fiber Distributed Data Interface (FDDI) NICs, and others, or any combination of the preceding. Of course, the use of particular types of NICs 27 depends on the types of existing LAN and backbone environments to which switch gateway unit 21 provides access. It is recognized that multiple LANs may be connected to a switch gateway unit 21. ATM NIC 29 allows switch gateway unit 21 to connect via an ATM link to a basic switching unit 1. Of course, a NIC 27 may also be an ATM NIC to provide a connection between switch gateway unit 21 and another switch gateway unit as well.

In addition to basic switching units and switch gateway units, the present system may also include high performance host computers, workstations, or servers that are appropriately equipped. In particular, a subset of the system software can be installed on a host computer, workstation, or server equipped with an appropriate ATM NIC to enable a host to connect directly to a basic switching unit.

As shown in Fig. 1c, a switching agent 901 according to yet another embodiment of the present invention, is a computer equipped with multiple network adaptors or NICs 903 for connection of existing LAN and backbone environments, an ATM NIC 905 for connection to a basic switching unit 1, and appropriate system software 907 that enables switching agent 901 to forward packets per instructions from a basic switching unit 1. Switching agent 901 serves as an access device to enable connection of existing LAN and backbone environments to at least one basic switching unit. Accordingly, NICs 903 may be of the same or different types, such

as for example 10BaseT Ethernet NICs, 100BaseT Ethernet NICs, FDDI NICs, and others, or any combination of the preceding. Of course, the use of particular types of NICs 903 depends on the types of existing LAN and backbone environments to which switching agent 901 provides access. It is recognized that multiple LANs may be connected to switching agent 901. ATM NIC 905 allows switching agent 901 to connect via an ATM link to a basic switching unit 1. Of course, NIC 905 is appropriately selected based on the specific switching engine technology, ATM in the present specific embodiment, utilized in basic switching unit 1.

Basic switching units, switch gateway units, switching agents, and system software allow users to build flexible IP network topologies targeted at the workgroup, campus, and WAN environments for high performance, scaleable solution to current campus backbone congestion problems. Using the present system, various network configurations may be implemented to provide end-to-end seamless IP traffic flow, with the network configurations featuring high bandwidth, high throughput, and component interoperability. Figs. 2a-2c illustrate a few of the many network configurations possible according to the present invention. Of course, Figs. 2a-2c are merely exemplary configurations and many alternate configurations are possible.

Fig. 2a shows a simplified diagram of a campus LAN configuration in which basic switching unit 1 serves as the centralized IP packet-forwarding device for the entire campus network with several switch gateway units 21 enabling connectivity to existing LANs. Basic switching unit 1 is connected to a server farm which includes three servers 31_n (where n = 1 to 3). Each server 31_n is equipped with a subset of the system software and an ATM NIC to enable connection to basic switching unit 1 via corresponding ATM links 33_n (where n = 1 to 3), which are OC-3 (155 Mbps) links. Having the servers attached directly to basic switching unit 1 over high speed ATM links operates to boost packet throughput for the frequently accessed servers. Basic switching unit 1 also connects to three switch gateway units 21 via corresponding ATM links 33_n (where n = 4 to 6), also OC-3 links. A first switch gateway unit 21 connected to basic switching unit 1 via link 33_4 also connects to a LAN backbone 35_1, which may be some type of Ethernet or FDDI, via an appropriate link 39_1. LAN backbone 35_1 connects to PCs, terminals, or

workstations 41 via the appropriate NICs 43. Similarly, second and third switch gateway units 21, connected to basic switching unit 1 via links 33_5 and 33_6 respectively, also connect to LAN backbones 35_2 and 35_3 respectively via Ethernet or FDDI links 39_2 and 39_3. The configuration of Fig. 2a therefore enables users connected to different LANs to communicate using seamless IP traffic flow without congestion in accordance with the present invention.

As another example, Fig. 2b shows a simplified diagram of a workgroup configuration. Fig. 2b illustrates a high performance workgroup environment in which several host computers 45 are connected via ATM links 33_m to multiple basic switching units 1, which connect to a switch gateway unit 21 that connects to a LAN 35 with user devices 41. In this configuration, a first basic switching unit 1 connects to a second basic switching unit 1 via ATM link 33_1 (155 Mbps). Multiple host computers 45 connect to the first basic switching unit 1 via respective 155 Mbps ATM links 33_x (where $x = 2$ to 5) through respective ATM NICs 47. In addition, multiple host computers 45 connect to the second basic switching unit 1 via respective 25 Mbps ATM links 33_y (where $y = 8$ to 10) through respective ATM NICs 49. As discussed above, host computers 45 equipped with ATM NICs are installed with a subset of the system software, enabling the TCP/IP hosts to connect directly to a basic switching unit. The first and second basic switching units 1 connect to switch gateway unit 21 via ATM links 33_6 (155 Mbps) and 33_7 (25 Mbps) respectively. Connection of the first and second basic switching units 1 to switch gateway unit 21 via an Ethernet or FDDI link 39 enables users of host computers 45 to communicate with users devices 41 attached to LAN 35. User devices 41 may be PCs, terminals, or workstations having appropriate NICs 43 to connect to any Ethernet or FDDI LAN 35. The workgroup of host computers is thereby seamlessly integrated with the rest of the campus network.

As still another example, Fig. 2c shows a simplified diagram of a simple configuration utilizing a basic switching unit 1; several switching agents 911, 913, and 915; and a system node 916 (e.g., another basic switching unit, switch gateway unit, or host). Of course, other configurations may involve additional system nodes and other combinations as desired. Fig. 2c illustrates several

switching agents 911, 913, and 915, each agent having respective interfaces to various Ethernet LANs 917_n (where n ranges from 1 to 6 in this specific example), each having connected user devices (not shown), and each agent being connected via ATM links 919_m (where m ranges from 1 to 3 in this specific example) to basic switching unit 1, which includes a switch controller 921 connected by an ATM link 923 to a switching engine 925. Of course, LANs 917_n may be FDDI, 10BaseT or 100BaseT Ethernet, Gigabit Ethernet, other type of network, or any combination of the types of networks. User devices connected to LANs 917_n may be PCs, terminals, printers, servers, workstations, etc. having appropriate NICs to connect to LANs 917_n. System node 916 is attached to the switching engine 925 of basic switching unit 1 via ATM link 919_4.

In general, switch controller 921 in Fig. 2c controls the switching agents by conditioning their respective interfaces (for the transmission and reception of packets) and by directing the switching agents in how to handle packets received in specific flows of specific flow types. The specific flow types, as well as the specific flow, may be created by switch controller 921 via operation of the IFMP-C protocol. As mentioned above, switch controller 921 is attached to a link layer switch (such as ATM switch 925), which in turn may be attached to switching agents (such as 911, 913, 915) and/or to another system node 916. During initialization, switch controller 921 sends IFMP-C packets to the switching agents, allowing switch controller 921 to learn the specific configuration (in terms of installed network interfaces, etc.) of each switching agent. Switch controller 921 then conditions one or more of the network interfaces 917_n attached to the switching agents to start receiving packets. Switch controller 921 also sets up the packet processing in the switching agent to transmit certain received packets to switch controller 921 while other received packets may be dropped (e.g., if they are received for protocols not being processed by switch controller 921). If switch controller 921 detects that a flow may be handled by a switching agent without intervention by switch controller 921, then switching controller 921 uses IFMP-C to direct that switching agent to handle the packet (e.g., drop a packet, forward the packet out one or more interfaces using one or more different output formats or using different classes of service to forward packets locally). Associated with

forwarding a packet is a transformation to apply to the packet (e.g., decrementing the Time to Live in the packet, updating IP header checksums, header managing for different flow type formats, etc.). Further details regarding the interoperation of switching agents, the switching node and the switch controller (such as shown in the configuration shown in Fig. 2c) are described below.

According to the present invention, the system adds complete IP routing functionality on top of ATM (or alternative technology in other embodiments) switching hardware by using the system software, instead of any existing ATM switch control software, to control the ATM switch. Therefore, the present system is capable of moving between network layer IP routing when needed and datalink layer switching when possible in order to create high speed and capacity packet transmission in an efficient manner without the problem of router bottlenecks.

Using the Ipsilon Flow Management Protocol (IFMP), which is described in further detail later, the system software enables a system node (such as a basic switching unit, switch gateway unit, or host computer/server/workstation) to classify IP packets as belonging to a "flow" of similar packets based on certain common characteristics. A flow is a sequence of packets sent from a particular source to a particular (unicast or multicast) destination that are related in terms of their routing and any local handling policy they may require. The present invention efficiently permits different types of flows to be handled differently, depending on the type of flow. Some types of flows may be handled by mapping them into individual ATM connections using the ATM switching engine to perform high speed switching of the packets. Flows such as for example those carrying real-time traffic, those with quality of service requirements, or those likely to have a long holding time, may be configured to be switched whenever possible. Other types of flows, such as for example short duration flows or database queries, are handled by connectionless IP routing. A particular flow of packets may be associated with a particular ATM label (i.e., an ATM virtual path identifier (VPI) and virtual channel identifier (VCI)). It is assumed that virtual channels are unidirectional so an ATM label of an incoming direction of each link is owned by the input port to which it is connected. Each direction of transmission on a link is treated separately. Of

course, flows travelling in each direction are handled by the system separately but in a similar manner.

Flow classification is a local policy decision. When an IP packet is received by a system node, the system node transmits the IP packet via the default channel. The node also classifies the IP packet as belonging to a particular flow, and accordingly decides whether future packets belonging to the same flow should preferably be switched directly in the ATM switching engine or continue to be forwarded hop-by-hop by the router software in the node. If a decision to switch a flow of packets is made, the flow must first be labelled. To label a flow, the node selects for that flow an available label (VPI/VCI) of the input port on which the packet was received. The node which has made the decision to label the flow then stores the label, flow identifier, and a lifetime, and then sends an IFMP REDIRECT message upstream to the previous node from which the packet came. The flow identifier contains the set of header fields that characterize the flow. The lifetime specifies the length of time for which the redirection is valid. Unless the flow state is refreshed, the association between the flow and label is deleted upon the expiration of the lifetime. Expiration of the lifetime before the flow state is refreshed results in further packets belonging to the flow to be transmitted on the default forwarding channel between the adjacent nodes. A flow state is refreshed by sending upstream a REDIRECT message having the same label and flow identifier as the original and having another lifetime. The REDIRECT message requests the upstream node to transmit all further packets that have matching characteristics to those identified in the flow identifier via the virtual channel specified by the label. The redirection decision is also a local decision handled by the upstream node, whereas the flow classification decision is a local decision handled by the downstream node. Accordingly, even if a downstream node requests redirection of a particular flow of packets, the upstream node may decide to accept or ignore the request for redirection. In addition, REDIRECT messages are not acknowledged. Rather, the first packet arriving on the new virtual channel serves to indicate that the redirection request has been accepted.

The system software also uses different encapsulations for the transmission of IP packets that belong to labelled flows on an ATM data link,

depending on the different flow type of the flows. In the present embodiment, four types of encapsulations are used.

In addition to IFMP, the system software utilizes another protocol, General Switch Management Protocol (GSMP), to establish communication over the ATM link between the switch controller and ATM hardware switching engine of a basic switching unit of the system and thereby enable layer 2 switching when possible and layer 3 IP routing and packet forwarding when necessary. In particular, GSMP is a general purpose, asymmetric protocol to control an ATM switch. That is, the switch controller acts as the master with the ATM switch as the slave. GSMP runs on a virtual channel established at initialization across the ATM link between the switch controller and the ATM switch. A single switch controller may use multiple instantiations of GSMP over separate virtual channels to control multiple ATM switches. Also included in GSMP is a GSMP adjacency protocol, which is used to synchronize state across the ATM link between the switch controller and the ATM switch, to discover the identity of the entity at the other end of the link, and to detect changes in the identity of that entity.

GSMP allows the switch controller to establish and release connections across the ATM switch, add and delete leaves on a point-to-multipoint connection, manage switch ports, request configuration information, and request statistics. GSMP also allows the ATM switch to inform the switch controller of events such as a link going down.

A switch is assumed to contain multiple ports, where each port is a combination of an input port and an output port. ATM cells arrive at the ATM switch from an external communication link on incoming virtual channels at an input port, and depart from the ATM switch to an external communication link on outgoing virtual channels from an output port. As mentioned earlier, virtual channels on a port or link are referenced by their VPI/VCI. A virtual channel connection across an ATM switch is formed by connecting an incoming virtual channel (or root) to one or more outgoing virtual channels (or branches). Virtual channel connections are referenced by the input port on which they arrive and the VPI/VCI of their incoming virtual channel. In the switch, each port has a hardware

look-up table indexed by the VPI/VCI of the incoming ATM cell, and entries in the tables are controlled by a local control processor in the switch.

For GSMP, each virtual channel connection may be established with a certain quality of service (QOS), by assigning it a priority when it is established. For virtual channel connections that share the same output port, an ATM cell on a connection with a higher priority would be more likely to depart the switch than an ATM cell on a connection with a lower priority, if they are both in the switch at the same time. The number of priorities each port of the switch supports is obtained from a port configuration message. It is recognized that different switches may support multicast in different ways. For example, the switch may have limits on numbers of branches for a multicast connection, limits on the number of multicast connections supported, limits on the number of different VPI/VCI values assignable to output branches of a multicast connection, and/or support only a single branch of a particular multicast connection on the same output port. Failure codes may be specified accordingly as required.

The switch assigns 32-bit port numbers to describe the switch ports. The port number may be structured into sub-fields relating to the physical structure of the switch (e.g., shelf, slot, port). Each switch port also maintains a port session number assigned by the switch. The port session number of a port remains the same while the port is continuously up. However, if a port returns to the up state after it has been down or unavailable or after a power cycle, the port session number of the port will change. Port session numbers are assigned using some form of random number, and allow the switch controller to detect link failures and keep state synchronized.

In addition to IFMP and GSMP, the system software in some embodiments also utilizes another protocol, Ipsilon Flow Management Protocol for Clients (IFMP-C), described in further detail below, to establish communication over the link between the switch controller of a basic switching unit and a switching agent to thereby distribute layer 3 packet forwarding to switching agents when desired. In particular, IFMP-C is a general purpose, asymmetric protocol to control a switching agent. That is, the switch controller acts as the master with the switching agent as the slave. With the use of IFMP-C, the interfaces on the

switching agent look like interfaces locally attached to the switch controller, so that the switch controller/switching agent externally appears to be like a system node. Generally, IFMP-C runs on a virtual channel established at initialization across the link between the switch controller and the switching agent. A single switch controller may use multiple instantiations of IFMP-C over separate virtual channels to control multiple switching agents. At system startup, the switch controller starts an IFMP-C listener on each ATM interface (the listener is attached the default VCI of the ATM interface) attached to the switch controller, and the switching agent begins sending period SYN messages on the default VCI. When the switch controller receives the SYN message from the switching agent, the switch controller starts the IFMP-C adjacency protocol, which is included in the IFMP-C protocol. Used by each side of the link, the IFMP-C adjacency protocol is used to synchronize state across the link between the switch controller and the switching agent, to discover the identity of the entity at the other end of the link, and to detect changes in the identity of that entity. When the IFMP-C adjacency protocol has established each side of the link to synchronize with the other, each side of the link has an instance number that identifies the other side of the link.

After completing synchronization, IFMP-C allows the switch controller to determine what ports or interfaces (and their attributes) are available on the switching agent, and to configure each interface so that it can be used to forward packets. A switching agent is assumed to contain multiple ports or interfaces, where each interface or port is a combination of an input port and an output port. Once the interfaces are determined and configured, IFMP-C is used to create, modify, and delete forwarding branches. Each forwarding branch consists of input data and output data. In the switching agent, each interface has a hardware look-up table indexed by the input data/output data of the incoming packet, and entries in the tables are controlled by a local control processor in the switching agent. The input data includes several pieces or components (such as input interface, precedence, input flags, key data, and key mask, according to a specific embodiment) of information, with each piece of information contributing to the input information. If any components of the input data vary, then the packet is considered to have a different forwarding input entry. The output data includes

several pieces or components (such as output interface, remove length, transform, transform data, header data, quality of service type, and quality of service data, according to a specific embodiment) that describe how packets having matching input data should be forwarded. It is possible for an input entry to have more than one output entry. When a packet arrives on an interface of the switching agent, the switching agent searches through the input entries associated with the input interface. The entries may be searched from the lowest precedence to the highest. When a matching input entry is found, the information on the output branches is used to forward the packet.

In IFMP-C, management of link level hardware (for example, opening virtual channels and adding hardware address filters on Ethernet) is left to the switching agent. If the input key mask includes bits of the link level address, the switching agent should ensure that it will receive those addresses. If the mask does not include link level addressing information, then the switching agent should not adjust the filter. The switching agent may thus control the link level filtering in the manner most efficient for its hardware, and the switch controller must include enough link level information in the key to properly filter packets. The switch controller manages the state of the switching agent for the promiscuous and multicast promiscuous modes, so that the switching agent does not attempt to inappropriately optimize the code path beyond the behavior desired.

IFMP, GSMP, and IFMP-C are described in further detail below, in accordance with a specific embodiment of the present invention.

II. System Hardware

A. Controller Hardware

Fig. 3 is a system block diagram of a typical computer system 51 that may be used as switch controller 5 in a basic switching unit 1 (as shown in Fig. 1a) to execute the system software of the present invention. Fig. 3 also illustrates an example of the computer system that may be used as switch gateway controller 23 in a switch gateway unit 21 (as shown in Fig. 1b) to execute the system software of the present invention, as well as serving as an example of a typical computer which may be used as a host computer/server/workstation loaded with a subset of the

system software. Of course, it is recognized that other elements such as a monitor, screen, and keyboard are added for the host. As shown in Fig. 3, computer system 51 includes subsystems such as a central processor 69, system memory 71, I/O controller 73, fixed disk 79, network interface 81, and read-only memory (ROM) 83. Of course, the computer system 51 optionally includes monitor 53, keyboard 59, display adapter 75, and removable disk 77, for the host. Arrows such as 85 represent the system bus architecture of computer system 51. However, these arrows are illustrative of any interconnection scheme serving to link the subsystems. For example, a local bus could be utilized to connect central processor 69 to system memory 71 and ROM 83. Other computer systems suitable for use with the present invention may include additional or fewer subsystems. For example, another computer system could include more than one processor 69 (i.e., a multi-processor system) or a cache memory.

In an embodiment of the invention, the computer used as the switch controller is a standard Intel-based central processing unit (CPU) machine equipped with a standard peripheral component interconnect (PCI) bus, as well as with an ATM network adapter or network interface card (NIC). The computer is connected to the ATM switch via a 155 Megabits per second (Mbps) ATM link using the ATM NIC. In this embodiment, the system software is installed on fixed disk 79 which is the hard drive of the computer. As recognized by those of ordinary skill in the art, the system software may be stored on a CD-ROM, floppy disk, tape, or other tangible media that stores computer readable code.

Computer system 51 shown in Fig. 3 is but an example of a computer system suitable for use (as the switch controller of a basic switching unit, as the switch gateway controller of a switch gateway unit, or as a host computer/server/workstation) with the present invention. Further, Fig. 3 illustrates an example of a computer system installed with at least a subset of the system software (to provide for IFMP-C operability) that may be used as a switching agent 901 (as shown in Fig. 1c). It should be recognized that system software for routing protocols need not be installed on a computer system used as a switching agent 901, and therefore this subset of the system software may be run on an embedded device. Accordingly, fixed disk 79 may be omitted from a computer system used as a

switching agent 901, thereby resulting in lower equipment costs for some networks which might use switching agents 901 in lieu of switch gateway units. Other configurations of subsystems suitable for use with the present invention will be readily apparent to one of ordinary skill in the art. In addition, switch gateway unit may be equipped with multiple other NICs to enable connection to various types of LANs. Other NICs or alternative adaptors for different types of LAN backbones may be utilized in switch gateway unit. For example, SMC 10M/100M Ethernet NIC or FDDI NIC may be used.

Without in any way limiting the scope of the invention, Table 1 provides a list of commercially available components which are useful in operation of the controller, according to the above embodiments. It will be apparent to those of skill in the art that the components listed in Table 1 are merely representative of those which may be used in association with the inventions herein and are provided for the purpose of facilitating assembly of a device in accordance with one particular embodiment of the invention. A wide variety of components readily known to those of skill in the art could readily be substituted or functionality could be combined or separated.

Table 1

Controller Components

Microprocessor	Intel Pentium 133 MHz processor
System memory	16Mbyte RAM/256K cache memory
Motherboard	Intel Endeavor motherboard
ATM NIC	Zeitnet PCI ATM NIC (155 Mbps)
Fixed or Hard disk	500Mbyte IDE disk
Drives	standard floppy, CD-ROM drive
Power supply	standard power supply
Chassis	standard chassis

B. Switching Hardware

As discussed above, the ATM switch hardware provides the switching engine of a basic switching unit. The ATM switching engine utilizes vendor-independent ATM switching hardware. However, the ATM switching engine

according to the present invention does not rely on any of its usual connection-oriented ATM routing and signaling software (SSCOP, Q.2931, UNI 3.0/3.1, and P-NNI). Rather, any ATM protocols and software are completely discarded, and a basic switching unit relies on the system software to control the ATM switching engine. The system software is described in detail later.

Separately available ATM components may be assembled into a typical ATM switch architecture. For example, Fig. 5 is a general block diagram of an architecture of an ATM switch 3 (the example shows a 16-port switch) that may be used as the switching hardware engine of a basic switching unit according to an embodiment of the present invention. However, commercially available ATM switches also may operate as the switching engine of the basic switching unit according to other embodiments of the present invention. The main functional components of switching hardware 3 include a switch core, a microcontroller complex, and a transceiver subassembly. Generally, the switch core performs the layer 2 switching, the microcontroller complex provides the system control for the ATM switch, and the transceiver subassembly provides for the interface and basic transmission and reception of signals from the physical layer. In the present example, the switch core is based on the MMC Networks ATMS 2000 ATM Switch Chip Set which includes White chip 100, Grey chip 102, MBUF chips 104, Port Interface Device (PIF) chips 106, and common data memory 108. The switch core also may optionally include VC Activity Detector 110, and Early Packet Discard function 112. Packet counters also are included but not shown. White chip 100 provides configuration control and status. In addition to communicating with White chip 100 for status and control, Grey chip 102 is responsible for direct addressing and data transfer with the switch tables. MBUF chips 104 are responsible for movement of cell traffic between PIF chips 106 and the common data memory 108. Common data memory 108 is used as cell buffering within the switch. PIF chips 106 manage transfer of data between the MBUF chips to and from the switch port hardware. VC Activity Detector 110 which includes a memory element provides information on every active virtual channel. Early Packet Discard 112 provides the ability to discard certain ATM cells as needed. Packet counters provide the switch with the ability to count all packets passing all input and output ports. Buses 114,

115, 116, 117, and 118 provide the interface between the various components of the switch. The microcontroller complex includes a central processing unit (CPU) 130, dynamic random access memory (DRAM) 132, read only memory (ROM) 134, flash memory 136, DRAM controller 138, Dual Universal Asynchronous Receiver-Transmitter (DUART) ports 140 and 142, and external timer 144. CPU 130 acts as the microcontroller. ROM 134 acts as the local boot ROM and includes the entire switch code image, basic low-level operation system functionality, and diagnostics. DRAM 132 provides conventional random access memory functions, and DRAM controller 138 (which may be implemented by a field programmable gate array (FPGA) device or the like) provides refresh control for DRAM 132. Flash memory 136 is accessible by the microcontroller for hardware revision control, serial number identification, and various control codes for manufacturability and tracking. DUART Ports 140 and 142 are provided as interfaces to communications resources for diagnostic, monitoring, and other purposes. External timer 144 interrupts CPU 130 as required. Transceiver subassembly includes physical interface devices 146, located between PIF chips 106 and physical transceivers (not shown). Interface devices 146 perform processing of the data stream, and implement the ATM physical layer. Of course, the components of the switch may be on a printed circuit board that may reside on a rack for mounting or for setting on a desktop, depending on the chassis that may be used.

Without in any way limiting the scope of the invention, Table 2 provides a list of commercially available components which are useful in operation of the switching engine, according to the above embodiments. It will be apparent to those of skill in the art that the components listed in Table 2 are merely representative of those which may be used in association with the inventions herein and are provided for the purpose of facilitating assembly of a device in accordance with a particular embodiment of the invention. A wide variety of components or available switches readily known to those of skill in the art could readily be substituted or functionality could be combined or separated. Of course, as previously mentioned, switching engines utilizing technologies (such as frame relay, fast packet switching, or Gigabit Ethernet) other than ATM would utilize appropriate components.

Table 2

Switch Components

SWITCH CORE	
Core chip set	MMC Networks ATMS 2000 ATM Switch Chip Set (White chip, Grey chip, MBUF chips, PIF chips)
Common data memory	standard memory modules
Packet counters	standard counters
MICROCONTROLLER COMPLEX	
CPU	Intel 960CA/CF/HX
DRAM	standard DRAM modules
ROM	standard ROM
Flash memory	standard flash memory
DRAM controller	standard FPGA, ASIC, etc.
DUART	16552 DUART
External timer	standard timer
TRANSCEIVER SUBASSEMBLY	
Physical interface	PMC-Sierra PM5346

III. System Software Functionality

As generally described above, IFMP is a protocol for instructing an adjacent node to attached a layer 2 "label" to a specified "flow" of packets. A flow is a sequence of packets sent from a particular source to a particular destination(s) that are related in terms of their routing and logical handling policy required. The label specifies a virtual channel and allows cached routing information for that flow to be efficiently accessed. The label also allows further packets belonging to the specified flow to be switched at layer 2 rather than routed at layer 3. That is, if both upstream and downstream links redirect a flow at a particular node in the network, that particular node may switch the flow at the datalink layer, rather than route and forward the flow at the network layer.

Fig. 5a is a simplified diagrams generally illustrating the initialization procedure in each system node according to an embodiment of the present invention. Upon system startup at step 160, each system node establishes default virtual channels on all ports in step 162. Then at step 164 each system node waits for packets to arrive on any port.

Fig. 5b is a simplified diagram that generally illustrates the operation of a system node dynamically shifting between layer 3 routing and layer 2 switching according to the present invention. After initialization, a packet arrives on a port of the system node at step 166. If the packet is received on a default virtual channel (step 168), the system node performs a flow classification on the packet at step 170. Flow classification involves determining whether the packet belongs to a type of flow. At step 172, the system node determines whether that flow to which the packet belongs should preferably be switched. If the system node determines that the flow should be switched, the system node labels the flow in step 174 then proceeds to forward the packet in step 176. After forwarding the packet, the system node waits for a packet to arrive in step 182. Once a packet arrives, the system node returns to step 166. If the system node determines at step 168 that the packet did not arrive on the default virtual channel, the system node does not perform flow classification at step 170 on the packet. When a packet arrives on an alternate virtual channel, the packet belongs to a flow that has already been labelled. Accordingly, if the flow is also labelled downstream (step 178), the system node switches the flow in step 180. Switching the flow involves making a connection within the switch between the label of the upstream link and the label of the downstream link. After switching the flow in step 180, the system node at step 176 forwards the packet downstream. If the flow is not labelled downstream (step 178), the system node does not switch the flow but rather forwards the packet downstream in step 176. Of course, it is recognized that only a system node that is a basic switching unit performs step 180. Other system nodes (e.g., switch gateway unit or host) operate as shown in Fig. 5b but do not perform step 180 since the result of step 178 is no for a switch gateway unit or a host (as these types of system nodes have no downstream link).

Fig. 5c and 5d are simplified diagrams that generally illustrate the operation of a switch controller and a switching agent attached to the switch controller via a communication link, respectively, according to the present invention. It is noted that a switching agent generally follows the initialization procedure illustrated by Fig. 5a. Fig. 5c generally illustrates the procedure at a switching agent when a packet arrives (step 1600) on one of its interfaces after initialization is completed. If the packet is not received on a default virtual channel (determined in step 1602), then the switching agent accesses the tree bound to the specified channel at step 1604. When a packet does not arrive on a default channel, the packet belongs to a flow that has already been labelled and the flow has been switched. The switching agent proceeds to forward the packet (step 1606) accordingly and then waits for another packet to arrive (step 1608). However, if the packet is received on the default virtual channel (determined in step 1602), then the switching agent searches its branch table for a matching input branch in step 1610. If a matching input branch is not found (in step 1612), the switching agent sends the packet to the switch controller in step 1614 and waits for another packet (step 1616). If a matching input branch is found (in step 1612), the switching agent forwards the packet as specified in step 1618. Then the switching agent checks if "fall through" mode is specified for the packet (step 1620). As discussed later, fall through mode indicates that the switching agent should continue the search in the branch table for a matching input branch at the next precedence level that matches this input branch entry after the packet is transmitted. If the fall through mode is not specified (step 1620), then the switching agent simply waits for the next packet to arrive (step 1622). However, if the fall through mode is specified (step 1620), then the switching agent continues to search in the branch table for a matching input branch at the next precedence level (step 1624). From step 1624, the switching agent determines whether the matching input branch at the next precedence level is found (step 1626). If it is not found, then the switching agent waits for the arrival of the next packet (step 1622). However, if it is found, then the switching agent proceeds from step 1626 to forward the packet as specified (step 1618), where the procedure continues from step 1620.

Fig. 5d generally illustrates the procedure at a switch controller (to which at least one switching agent may be attached via a communication link, for example, using the switching engine of the switch controller) when a packet arrives (step 1650) from a switching agent on one of its interfaces on a default channel, after initialization is completed. That is, Fig. 5d illustrates the procedure at the switch controller upon a packet being sent to the switch controller (step 1614 of Fig. 5c). After the packet arrives (step 1650) from the switching agent, the switch controller performs a flow classification on the packet at step 1652. As mentioned above, flow classification involves determining whether the packet belongs to a type of flow. From step 1652, the switch controller determines in step 1654 whether the flow to which the packet belongs should be switched. If the switch controller determines in step 1654 that the flow should not be switched, the switch controller does not switch the flow but simply forwards the packet (step 1656) and then waits for the next packet (step 1658). If the switch controller determines in step 1654 that the flow should preferably be switched, then the switch controller labels the flow in step 1660 and proceeds to forward the packet (step 1656) and wait for the next packet (step 1658).

Fig. 6a is a diagram generally illustrating the steps involved in labelling a flow in the upstream link of a system node (or a switching node), such as shown by label flow step 174 of Fig. 5b. For a system node that is a switch gateway unit or a host, the system node labels a flow as shown in steps 190, 192, 200 and 202 of Fig. 6a. When the label flow step begins (step 190), the system node selects a free label x on the upstream link in step 192. The system node then sends an IFMP REDIRECT message on the upstream link in step 200 (as indicated by dotted line 193). The system node then forwards the packet in step 202. For a system node that is a basic switching unit, labelling a flow is also illustrated by steps 194, 196, and 198. When the label flow step begins (step 190), the basic switching unit selects a free label x on the upstream link in step 192. The switch controller of basic switching unit then selects a temporary label x' on the control port of the switch controller in step 194. At step 196, the switch controller then sends to the hardware switching engine a GSMP message to map label x on the upstream link to label x' on the control port. The switch controller then waits in

step 198 until a GSMP acknowledge message is received from the hardware switching engine that indicates that the mapping is successful. Upon receiving acknowledgement, the basic switching unit sends an IFMP REDIRECT message on the upstream link in step 200. After step 200, the system node returns to step 176 as shown in Fig. 56.

Fig. 6b is a diagram generally illustrating the steps involved in switching a flow in a basic switching unit, such as shown by switch flow step 180 of Fig. 5b. As mentioned above, only system nodes that are basic switching units may perform the switch flow step. When the switch flow procedure starts in step 210, the switch controller in the basic switching unit sends at step 212 a GSMP message to map label x on the upstream link to the label y on the downstream link. Label y is the label which the node downstream to the basic switching unit has assigned to the flow. Of course, this downstream node has labelled the flow in the manner specified by Figs. 5b and 6a, with the free label y being selected in step 192. After step 212, the switch controller in the basic switching unit waits in step 214 for a GSMP acknowledge message from a hardware switching engine in basic switching unit to indicate that the mapping is successful. The flow is thereby switched in layer 2 entirely within the hardware switching engine in the basic switching unit. Then the basic switching unit proceeds to forward the packet in step 176.

Fig. 6c is a diagram generally illustrating the steps involved in forwarding a packet in a system node, such as shown by forward packet step 176 of Fig. 5b. A system node at step 218 starts the forward packet procedure. If the flow to which the packet belongs is not labelled on the downstream link (step 220), then the system node sends the packet on the default virtual channel on the downstream link in step 222 and then goes to a wait state 224 to wait for arrival of packets. However, if the flow to which the packet belongs is labelled on the downstream link indicating that the system node previously received an IFMP REDIRECT message to label that flow for a lifetime, then the system node checks at step 226 if the lifetime for the redirection of that flow has expired. If the lifetime has not expired, then the system node sends the packet on the labelled virtual channel in the IFMP REDIRECT message at step 228 then goes to wait state

224. If the lifetime has expired, then the system node automatically deletes the flow redirection at step 230. The system node then proceeds to send the packet on the default channel (step 222) and returns to the wait state of step 182 as shown in Fig. 5b.

As described above, Figs. 6a-6c generally relate to the interoperation of system nodes (or switching nodes) without the involvement of switching agents. Figs. 6d-6e relate to the interoperation of switching nodes when at least one switching agent is attached to a basic switching unit, as described below.

Fig. 6d is a diagram generally illustrating the steps performed in the switch controller in labelling a flow for packets received from an attached source switching agent, such as shown by label flow step 1660 of Fig. 5d. Three scenarios are illustrated in Fig. 6d: when the flow of packets is desired to be sent to another interface on the source switching agent; when the flow of packets is desired to be sent to an interface on another attached switching agent, i.e., a destination switching agent; and when the flow of packets is desired to be sent to an interface on another attached system node (or switching node, such as another basic switching unit, a switch gateway unit, or a host).

As shown in Fig. 6d, if the flow of packets received from a source switching agent is desired to be sent to another interface on the same switching agent (as determined in step 1662), the switch controller (in step 1664) uses IFMP-C to condition the source switching agent to forward future packets received for the flow with the appropriate header and transformation out on the destination interface of that switching agent.

If the flow of packets received from a source switching agent is not desired to be sent to another interface on the same switching agent (as determined in step 1662), then it is determined in step 1666 if the flow of packets received from a source switching agent is desired to be sent to an interface on a destination switching agent. If so, the switch controller (in step 1668) selects a free label x on the upstream link between the switch controller and the source switching agent, and selects (in step 1670) a free label y on the downstream link between the switch controller and the destination switching agent. Then the switch controller uses GSMP to map x to y in step 1672. In step 1674, the switch controller uses IFMP-C

to condition the destination switching agent to forward out on the destination interface the future packets for the flow received on label y with the appropriate header and transformation. Then the switch controller (in step 1676) uses IFMP-C to condition the source switching agent to forward future packets of the flow with the appropriate header and transformation to label x.

If the flow of packets received from a source switching agent is not desired to be sent to an interface on a destination switching agent (as determined in step 1666), then the flow of packets received from the source switching agent is desired to be sent to an interface on another attached system node (or "switching node", such as another basic switching unit, a switch gateway unit, or a host). Then, the switch controller (in step 1680) selects a free label x on the upstream link between the switch controller and the source switching agent. In step 1682, the switch controller waits for a free label y on the downstream link to be chosen by the switching node and communicated via IFMP. Then, the switch controller uses GSMP to map x to y in step 1684. In step 1686, the switch controller uses IFMP-C to condition the source switching agent to forward future packets of the flow with the appropriate header and transformation to label x.

Fig. 6e is a diagram generally illustrating the steps performed in the switch controller in labelling a flow (starting from step 1700) for packets, which are received from an attached switching node and intended for an interface on an attached switching agent. When the flow of packets received from a source switching node is desired to be sent to an interface on a destination switching agent, the switch controller (in step 1702) selects a free label x on the upstream link between the switch controller and the source switching node, and selects (in step 1704) a free label y on the downstream link between the switch controller and the destination switching agent. Then the switch controller uses GSMP to map x to y in step 1706. In step 1708, the switch controller uses IFMP-C to condition the destination switching agent to forward out on the destination interface the future packets for the flow received on label y with the appropriate header and transformation. In step 1710, the switch controller uses IFMP to request the upstream switching node to transmit future packets of the flow to label x. Additional details of the general description above are described below.

A. IFMP & Flow Labelled Transmission on ATM Data Links

1. IFMP

The system software uses the Ipsilon Flow Management Protocol (IFMP) to enable a system node (such as a basic switching unit, switch gateway unit, or host computer/server/workstation) to classify IP packets as belonging to a flow of similar packets based on certain common characteristics. Flows are specified by a "flow identifier." The flow identifier for a particular flow gives the contents or values of the set of fields from the packet header that define the flow. The contents of the set of fields from the packet headers are the same in all packets belonging to that particular flow. Several "flow types" may be specified. Each flow type specifies the set of fields from the packet header that are used to identify the flow. For example, one flow type may specify the set of fields from the packet header that identify the flow as having packets carrying data between applications running on stations, while another flow type may specify the set of fields from the packet header that identify the flow as having packets carrying data between the stations.

In an embodiment of the present invention, three flow types are specified: Flow Type 0, Flow Type 1, and Flow Type 2. Of course, different or additional flow types also may be specified. Flow Type 0 is used to change the encapsulation of IP packets from the default encapsulation. The format of a flow identifier for Flow Type 0 is null and accordingly has a zero length. Flow Type 1 is a flow type that specifies the set of fields from the packet header that identify the flow as having packets carrying data between applications running on stations. Flow Type 1 is useful for flows having packets for protocols such as UDP and TCP in which the first four octets after the IP header specify a source port number and a destination port number that are used to indicate applications. A flow identifier for Flow Type 1 has a length of four 32-bit words. The format of a flow identifier for Flow Type 1, indicated as reference number 240 shown in Fig. 7a, includes (described in order of most significant bit (MSB) to least significant bit (LSB)) the Version, Internet Header Length (IHL), Type of Service, and Time to Live, and Protocol fields as the first word; the Source Address field as the second word; and the Destination Address field as the third word. These fields in the flow identifier

are from the header of the IP packet of Flow Type 1. The flow identifier for Flow Type 1 also includes the Source Port Number and the Destination Port Number fields (the first four octets in the IP packet after the IP header) as the fourth word. Flow Type 2 is a flow type that specifies the set of fields from the packet header that identify the flow as having packets carrying data between stations without specifying the applications running on the stations. A flow identifier for Flow Type 2 has a length of three 32-bit words. The format of a flow identifier for Flow Type 2, indicated by reference number 250 shown in Fig. 7b, includes the Version, Internet Header Length (IHL), Type of Service, Time to Live, Protocol, Source Address, and Destination Address fields from the header of the IP packet. The format of a flow identifier for Flow Type 2 is the same as that for Flow Type 1 without the fourth word. The hierarchical nature of the flow identifiers for the various flow types allows a most specific match operation to be performed on an IP packet to facilitate flow classification.

The present invention efficiently permits different types of flows to be handled differently, depending on the type of flow. Flows such as for example those carrying real-time traffic, those with quality of service requirements, or those likely to have a long holding time, may be configured to be switched whenever possible. Other types of flows, such as for example short duration flows or database queries, are handled by connectionless IP packet forwarding. In addition, each flow type also specifies an encapsulation that is to be used after this type of flow is redirected. Encapsulations for each flow type may be specified for different data link technologies. In the present embodiment, the system uses encapsulations for ATM data links, described in further detail later.

A particular flow of packets may be associated with a particular ATM label. According to the present embodiment, a label is a virtual path identifier and virtual channel identifier (VPI/VCI). A "range" of labels for a specific port is the set of labels (VPIs/VCIs) available for use at that port. It is assumed that virtual channels are unidirectional so a label of an incoming direction of each link is owned by the input port to which it is connected. Of course, for embodiments using other switching technologies such as frame relay, the data link connection identifier may

be used as the label. For embodiments using fast packet switching technology, the data link channel multiplex identifier may be used as the label.

As discussed above, flow classification is a local policy decision. When an IP packet is received by a system node, the system node transmits the IP packet via the default channel. The node also classifies the IP packet as belonging to a particular flow, and accordingly decides whether future packets belonging to the same flow should be switched directly in the ATM switching engine or continue to be forwarded hop-by-hop by the router software in the node. If a decision to switch a flow of packets is made, the node selects for that flow an available label (VPI/VCI) of the input port on which the packet was received. The node which has made the decision to switch the flow then stores the label, flow identifier, and a lifetime, and then sends an IFMP REDIRECT message upstream to the previous node from which the packet came. As discussed above, the flow identifier contains the set of header fields that characterize the flow. The lifetime specifies the length of time for which the redirection is valid. Unless the flow state is refreshed, the association between the flow and label should be deleted upon the expiration of the lifetime. Expiration of the lifetime before the flow state is refreshed results in further packets belonging to the flow to be transmitted on the default forwarding channel between the adjacent nodes.

A flow state is refreshed by sending upstream a REDIRECT message having the same label and flow identifier as the original and having another lifetime. The REDIRECT message requests the upstream node to transmit all further packets that have matching characteristics to those identified in the flow identifier via the virtual channel specified by the label. The redirection decision is also a local decision handled by the upstream node, whereas the flow classification decision is a local decision handled by the downstream node. Accordingly, even if a downstream node requests redirection of a particular flow of packets, the upstream node may decide to accept or ignore the request for redirection. In addition, REDIRECT messages are not acknowledged. Rather, the first packet arriving on the new virtual channel serves to indicate that the redirection request has been accepted.

In the present invention, IFMP of the system software includes an IFMP adjacency protocol and an IFMP redirection protocol. The IFMP adjacency

protocol allows a system node (host, basic switching unit, or switch gateway unit) to discover the identity of a system node at the other end of a link. Further, the IFMP adjacency protocol is used to synchronize state across the link, to detect when a system node at the other end of a link changes, and to exchange a list of IP addresses assigned to a link. Using the IFMP redirection protocol, the system may send REDIRECT messages across a link, only after the system has used the IFMP adjacency protocol to identify other system nodes at the other end of a link and to achieve state synchronization across a link. Any REDIRECT message received over a link that has not currently achieved state synchronization must be discarded. The IFMP adjacency protocol and IFMP redirection protocol are described in detail after the following detailed description of the operation of the system.

A specific example describing the flow classification and redirection of the present system, utilizing a LAN configuration such as that of Fig. 2a, is useful in illustrating advantages presented by the system. In particular, the example focuses on the interaction between the first and second gateway switch units 21 and basic switching unit 1 of Fig. 2a. At system startup, a default forwarding ATM virtual channel is established between the system software running on the controllers of basic switching unit 1 and of each of the neighboring nodes (in this example, first and second switch gateway units 21). When an IP packet is transmitted from LAN backbone 35_2 over the network layer link 39_1, the IP packet is received by the first switch gateway unit 21 via one of its appropriate LAN NICs. Then, the system software at first switch gateway unit 21 inspects the IP packet and then performs a default encapsulation of the IP packet contents for transmission via link 33_4 (established between the ATM NIC of switch gateway unit 21 and a selected port of the ATM switching hardware in basic switching unit 1) to basic switching unit 1. The ATM switching hardware then forwards the ATM cells to ATM NIC 9 in switch controller 5 which then reassembles the packet and forwards the IP datagram to the system software in switch controller for IP routing. The switch controller forwards the packet in the normal manner across the default forwarding channel initially established between basic switching unit 1 and second switch gateway unit 21 at startup. In addition, the switch controller in basic switching unit 1 performs a flow classification on the packet to determine whether future packets belonging to

the same flow should be switched directly in the ATM hardware or continue to be routed hop-by-hop by the system software. If the switch controller software decides locally that the flow should be switched, it selects a free label (label x) from the label space (label space is merely the range of VPI/VCI labels) of the input port (port i) on which the packet was received. The switch controller also selects a free label (label x') on its control port (the real or virtual port by which the switch controller is connected to the ATM switch). Using GSMP, the system software instructs the ATM switch to map label x on input port i to label x' on the control port c. When the switch returns a GSMP acknowledgement message to the switch controller, the switch controller sends an IFMP REDIRECT message upstream to the previous hop (in this example, the first switch gateway unit 21) from which the packet came. The REDIRECT message is simply a request from basic switching unit 1 to first switch gateway unit 21 to transmit all further packets with header fields matching those specified in the redirection message's flow identifier on the ATM virtual channel specified by the REDIRECT message's label. Unless the flow state is refreshed before the expiration of the REDIRECT message's lifetime, the association between the flow and the redirection message's label should be deleted, resulting in further packets in the flow being transmitted on the default forwarding channel (initially established at startup) between the first switch gateway unit 21 and basic switching unit 1.

If the first switch gateway unit 21 accepts the request made in the REDIRECT message sent by basic switching unit 1, the packets belonging to the flow will arrive at port c of switch controller with the ATM VPI/VCI label x'. The packets will continue to be reassembled and routed by the system software, but the process is speeded up as a result of the previous routing decision for the flow being cached and indexed by the label x' in the system software. Accordingly, it is seen that a flow may be labelled but not necessarily switched.

One of the important benefits of switching becomes evident in situations where the downstream node (in this example, the second switch gateway unit) also is involved in redirection for the same flow. When basic switching unit 1 routes the initial packet belonging to the flow to the second switch gateway unit 21 via the default forwarding channel between them, the downstream node (in this part

of the example, second switch gateway unit 21) reassembles the packet and forwards it in the normal manner. For the packet received at its port j, second switch gateway unit 21 also performs a flow classification and decides based upon its local policy expressed in a table whether to switch future packets belonging to the flow or to continue packet forwarding in the controller. If second switch gateway unit 21 decides that the future packets of the flow should be switched, it sends its own REDIRECT message (with a free label y on its port j, flow identifier, and lifetime) upstream to basic switching unit 1. Basic switching unit 1 may of course accept or ignore the request for redirection. When basic switching unit 1 decides to switch the flow, the system software in switch controller of basic switching unit 1 maps label x on port i to label y on port j. Thus, the traffic is no longer sent to the switch control processor but is switched directly to the required output port of the ATM switch hardware. Accordingly, all further traffic belonging to the flow may be switched entirely within the ATM switching hardware of basic switching unit 1. When a packet arrives from a port of the ATM switch of basic switching unit 1, second switch gateway unit 21 using its ATM NIC receives the packet over ATM link 33_5. Second switch gateway unit 21 then reassembles and sends the packet via one of its NICs over the link 39_2 to LAN 35_2. The user device 41 for which packet is intended receives it from LAN 35_2 via the user device's NIC 43.

When a system node (in this example, basic switching unit 1) accepts a REDIRECT message, it also changes the encapsulation used for the redirected flow. Rather than using the default encapsulation used for IP packets on the default forwarding channel, the system node may use a different type of encapsulation depending on the flow type. Basic switching unit 1 thus encapsulates the future packets belonging to the flow and transmits them on the specified virtual channel noted in label y. Some types of encapsulation may remove certain fields from the IP packet. When these fields are removed, the system node that issued the REDIRECT message stores the fields and associates the fields with the specified ATM virtual channel. In the case of the present example, if basic switching unit 1 accepts the REDIRECT message sent by second switch gateway unit 21, then basic switching unit 1 stores fields and associates the fields with the ATM virtual channel

specified by label y. Similarly, if first switch gateway unit 21 accepts the REDIRECT message sent by first switching unit 1, then first switch gateway unit 21 stores fields and associates the fields with the ATM virtual channel specified by label x. A complete packet may be reconstructed using the incoming label to access the stored fields. This approach provides a measure of security by for example preventing a user from establishing a switched flow to a permitted destination or service behind a fireball and then changing the IP packet header to gain access to a prohibited destination.

Each system node maintains a background refresh timer. When the background refresh timer expires, the state of every flow is examined. If a flow has received traffic since the last refresh period, the system node refreshes the state of that flow by sending a REDIRECT message upstream with the same label and flow identifier as the original REDIRECT message and a new lifetime. If the flow has received no traffic since the last refresh period, the system node removes the flow's cached state. A system node removes the flow's state by issuing an IFMP RECLAIM message upstream to reclaim the label for reuse. However, until the upstream node sends an IFMP RECLAIM ACK message which is received by the node issuing the IFMP RECLAIM message, the flow state is not deleted and the label may not be reused. An IFMP RECLAIM ACK message acknowledges release of the requested label. A system node determines if a flow has received traffic in two different ways, depending on whether the flow is switched or not. For flows that are labelled but not switched, the controller for the system node examines its own state to see whether the flow has received any traffic in the previous refresh period. For flows that are switched, the controller for the system node queries the ATM switch hardware using a GSMP message to see whether a specific channel has been active recently. Accordingly, in the present example, basic switching unit 1 monitors traffic for a flow if that particular flow is mapped from first switch gateway unit 21 to the control port of basic switching unit 1 or is mapped from first switch gateway unit 21 to second switch gateway unit 21 via the ATM switch in basic switching unit 1. If that flow has no recent traffic in the previous refresh period, basic switching unit will send the IFMP RECLAIM message and remove the flow state when an IFMP RECLAIM ACK message is received. Also, second

switch gateway unit 21 monitors traffic for a flow if that particular flow is mapped from the control port of basic switching unit 1 to second switch gateway unit 21. Additionally, a host computer/server/workstation equipped with the appropriate system software is also equipped with a background refresh timer. Monitoring traffic for any flow mapped to it, the host can send an IFMP RECLAIM message and remove a flow state upon receiving an IFMP RECLAIM ACK message.

As discussed above, the IFMP adjacency protocol is used to establish state synchronization, as well as identifying adjacent system nodes and exchanging IP addresses. For IFMP adjacency protocol purposes, a system node has three possible states for a particular link: SYNSENT (synchronization message sent), SYNRCVD (synchronization message received), ESTAB (synchronization established). State synchronization across a link (when a system node reaches the ESTAB state for a link) is required before the system may send any redirection messages using the IFMP redirection protocol.

Fig. 8a illustrates the structure of a generic IFMP adjacency protocol message 300. All IFMP adjacency protocol messages are encapsulated within an IP packet. Fig. 8b illustrates a generic IP packet (in its current version IPv4) with a variable length Data field into which an IFMP adjacency protocol message may be encapsulated. As an indication that the IP packet contains an IFMP message, the Protocol field in the IP header of the encapsulating IP packet must contain the decimal value 101. The Time to Live field in the header of the IP packet encapsulating the IFMP message is set to 1. Also, all IFMP adjacency protocol messages are sent to the limited broadcast IP Destination Address (255.255.255.255), using the address in the Destination Address field of the IP header. As seen in Fig. 8a, an IFMP adjacency protocol message 300 includes (described in order of MSB to LSB) the following fields: an 8-bit Version (302), an 8-bit Op Code (304), and a 16-bit Checksum (306) as the first 32-bit word; Sender Instance (308) as the second 32-bit word; Peer Instance (310) as the third 32-bit word; Peer Identity (312) as the fourth 32-bit word; Peer Next Sequence Number (314) as the fifth 32-bit word; and Address List (316) which is a field of a variable number of 32-bit words.

In an IFMP adjacency protocol message, Version field 302 specifies the version of the IFMP protocol which is currently in use (as other versions may evolve). Op Code 304 specifies the function of the IFMP adjacency protocol message. In the present embodiment, there are four possible Op Codes, i.e., functions of IFMP adjacency protocol messages: SYN (synchronization message, Op Code = 0), SYNACK (synchronization acknowledge message, Op Code = 1), RSTACK (reset acknowledge message, Op Code = 2), and ACK (acknowledge message, Op Code = 3). In each system node, a timer is required for the periodic generation of SYN, SYNACK, and ACK messages. In the present embodiment, the period of the timer is one second, but other periods may be specified. If the timer expires and the system node is in the SYNSENT state, the system node resets the timer and sends a SYN IFMP adjacency protocol message. If the timer expires and the system node is in the SYNRCVD state, the system node resets the timer and sends a SYNACK IFMP adjacency protocol message. If the timer expires and the system node is in the ESTAB state, the system node resets the timer and sends an ACK IFMP adjacency protocol message.

Checksum 306 is the 16-bit one's complement of the one's complement sum of: the source address, destination address and protocol fields from the IP packet encapsulating the IFMP adjacency protocol message, and the total length of the IFMP adjacency protocol message. Checksum 306 is used by the system for error control purposes.

In discussing IFMP, a "sender" is the system node which sends the IFMP message, and a "peer" is the system node to which the sender sends the IFMP message for a link.

In SYN, SYNACK, and ACK IFMP adjacency protocol messages, Sender Instance 308 is the sender's "instance number" for the link. Indicating a specific instance of a link, an instance number is a 32-bit non-zero number that is guaranteed to be unique within the recent past, and to change when the link or system node comes back after going down. Accordingly, each link has its own unique instance number. Sender Instance is used to detect when a link comes back after going down, or when the identity of a peer at the other end of the link changes. (Sender Instance 308 is used in a similar manner to the initial sequence

number (ISN) in TCP.) For a RSTACK IFMP adjacency protocol message, Sender Instance 308 is set to the value of the Peer Instance field 310 from the incoming message that caused the RSTACK message to be generated.

In SYN, SYNACK, and ACK IFMP adjacency protocol messages, Peer Instance field 310 is what the sender believes is the peer's current instance number for the link. If the sender does not know the peer's current instance number for the link, the Peer Instance field 310 will be set to zero. In an RSTACK IFMP adjacency protocol message, Peer Instance field 310 is set to the value of the Sender Instance field 308 from the incoming message that caused the RSTACK message to be generated.

For SYN, SYNACK, and ACK IFMP adjacency protocol messages, Peer Identity field 312 is the IP address of the peer that the sender of the message believes is at the other end of the link. The sender takes the IP address that is in the Source Address field of the IP header encapsulating the SYN or SYNACK message received by the sender, and uses that IP address in the Peer Identity field 312 of an IFMP adjacency protocol message it is sending. When the sender does not know the IP address of the peer at the other end of the link, Peer Identity field 312 is set to zero. For an RSTACK message, Peer Identity field 312 is set to the value of the IP address of the Source Address field from the IP header of the incoming IFMP adjacency protocol message that caused the RSTACK message to be generated.

Peer Next Sequence Number field 314 gives the value of the peer's Sequence Number field that the sender expects to arrive in the next IFMP redirection protocol message. If the value of the Peer Next Sequence Number 314 in an incoming IFMP adjacency protocol ACK message is greater than the value of one plus the value of the Sequence Number (from the last IFMP redirection protocol message transmitted out of the port on which the incoming IFMP adjacency protocol ACK message was received), then the link should be reset.

Address List field 316 is a list of one or more IP addresses that are assigned to the link by the sender of the IFMP adjacency protocol message. The list must have at least one entry which is identical to the Source Address of the IP

header of the IFMP adjacency protocol message. The contents of the list are not used by the IFMP but rather may be made available to the routing protocol.

Fig. 8c is a simplified diagram illustrating the operation of a system node upon receiving a packet with an incoming IFMP adjacency protocol message. After startup of the system, the system node receives a packet with an incoming IFMP adjacency protocol message (step 320). At step 322, the system node determines if the incoming IFMP adjacency protocol message is an RSTACK message. If the incoming IFMP adjacency protocol message is not an RSTACK message (e.g., a SYN, SYNACK, or ACK message), then the system node operates in the manner illustrated in the state diagram of Fig. 8d. If the incoming IFMP adjacency protocol message is an RSTACK message, then the system node checks at step 324 whether the Sender Instance and Source IP Address in the incoming RSTACK message matches the values stored from a previous message by the Update Peer Verifier operation for the port on which the incoming RSTACK message was received. For IFMP adjacency protocol, the Update Peer Verifier operation is defined as storing the Sender Instance and Source IP Address from a SYN or SYNACK message received from the peer on a particular port. If the values match from step 324, then the system node determines at step 326 whether the Peer Instance and Peer Identity in the incoming RSTACK message matches the values of Sender Instance and Source IP Address currently in use for all SYN, SYNACK, and ACK messages transmitted out of the port on which the incoming RSTACK message was received. If the values match from step 326, the system node determines at step 328 if the system node is in the SYNSENT state. If the system node is not in the SYNSENT state, the system node proceeds to reset the link at step 330. If the values do not match from step 324, or the values do not match from step 326, or the system node is in the SYNSENT state, then the system node discards the incoming RSTACK message at step 332 and waits for another packet to arrive. Accordingly when a RSTACK IFMP adjacency protocol message arrives at a system node, the system node resets the link, as indicated by steps 334, 336, 338, 340, and 342. In step 334, the system node generates a new instance number for the link. Then the system node in step 336 deletes the Peer Verifier (i.e., sets the stored values of Sender Instance and Source IP Address of the peer to

zero). At step 338, the system node sets the Sequence Number and Peer Next Sequence Number to zero. The system node then sends a SYN IFMP adjacency protocol message in step 340, and enters the SYNSENT state in step 342. The system node then receives another packet for processing.

Fig. 8d is a state diagram illustrating the operation of a sender system node when the incoming IFMP adjacency protocol message is not an RSTACK message. For the following description of Fig. 8d, condition "%B" is defined as: Sender Instance and Source IP Address in the incoming message matches the values stored from a previous message by the Update Peer Verifier operation for the port on which the incoming IFMP adjacency protocol message was received. Condition "%C" in Fig. 8d is defined as: the Peer Instance and Peer Identity in the incoming message matches the values of Sender Instance and Source IP Address currently in use for all SYN, SYNACK, and ACK messages transmitted out of the port on which the incoming IFMP adjacency protocol message was received. In Fig. 8d, condition "A" signifies that the sender system node receives an incoming SYNACK IFMP adjacency protocol message and that condition %C is met; condition "B" signifies that the sender system node receives an incoming SYNACK IFMP adjacency protocol message and that condition %C is not met; condition "C" signifies that the sender system node receives an incoming ACK IFMP adjacency protocol message and that conditions %B and %C are both met; and condition "D" signifies that the sender system node receives an incoming ACK IFMP adjacency protocol message and that conditions %B and %C are not both met.

If the sender is in the SYNSENT state 350 and receives an incoming SYN IFMP adjacency protocol message from a peer on the other end of a link, the sender performs an Update Peer Verifier operation and sends a SYNACK IFMP adjacency protocol message to the peer (indicated as step 352). Then the sender goes from SYNSENT state 350 to the SYNRCVD state 354. If the sender receives an incoming SYN IFMP adjacency protocol message while in SYNRCVD state 354, the sender in step 352 performs an Update Peer Verifier operation and sends a SYNACK IFMP adjacency protocol message to the peer, but remains in the SYNRCVD state 354. If the sender is in SYNRCVD state 354 and either condition B or condition D is met, then the sender sends an RSTACK IFMP adjacency

protocol message to the peer (indicated as step 356), and remains in SYNRCVD state 354. If the sender is in the SYNRCVD state 354 and condition C is met, then the sender sends an ACK IFMP adjacency protocol message to the peer (indicated as step 358), and moves to the ESTAB state 360. If the sender is in SYNRCVD state 354 and condition A is met, then the sender performs an Update Peer Identifier operation and sends an ACK IFMP adjacency protocol message to the peer (indicated as step 362), and moves to ESTAB state 360. The sender is and remains in ESTAB state 360, if the sender receives either a SYN or SYNACK IFMP adjacency protocol message or if condition C is met. If condition D is met while the sender is in ESTAB state 360, then the sender remains in ESTAB state 360 and sends a RSTACK IFMP adjacency protocol message (indicated as step 356). While in SYNSENT state 350, if either the sender receives an ACK IFMP adjacency protocol message or condition B is met, then the sender remains in SYNSENT state 350 and sends a RSTACK IFMP adjacency protocol message (step 356). If condition A is met when the sender is in SYNSENT state 350, then the sender performs an Update Peer Verifier operation and sends an ACK IFMP adjacency protocol message (step 362) and enters ESTAB state 360.

As discussed above, the IFMP redirection protocol is used to send redirection messages across a link, after the system has used the IFMP adjacency protocol to identify other system nodes at the other end of a link and to achieve state synchronization across a link. Any IFMP redirection message received over a link that has not currently achieved state synchronization must be discarded.

Fig. 9a illustrates the structure of a generic IFMP redirection protocol message 380. Like all IFMP adjacency protocol messages, all IFMP redirection protocol messages are encapsulated within an IP packet. Fig. 8b illustrates a generic IP packet (in its current version IPv4) with a variable length Data field into which an IFMP redirection protocol message may be encapsulated. As an indication that the IP packet contains an IFMP message, the Protocol field in the IP header of the encapsulating IP packet must contain the decimal value 101, and the Time to Live field in the header of the IP packet encapsulating the IFMP message is set to 1. An IFMP redirection protocol message is sent to the IP address of the peer at the other end of the link (the IP address being obtained from the

IFMP adjacency protocol), using the IP address in the Destination Address field of the IP header. As seen in Fig. 9a, an IFMP redirection protocol message 380 includes (described in order of MSB to LSB) the following fields: an 8-bit Version (382), an 8-bit Op Code (384), and a 16-bit Checksum (386) as the first 32-bit word; Sender Instance (388) as the second 32-bit word; Peer Instance (390) as the third 32-bit word; Sequence Number (392) as the fourth 32-bit word; and a Message Body (394) which is a field of a variable number of 32-bit words.

In an IFMP redirection protocol message, Version field 382 specifies the version of the IFMP protocol which is currently in use (as other versions may evolve). Op Code 384 specifies the function of the IFMP redirection protocol message. In the present embodiment, there are five possible Op Codes, i.e., functions of IFMP redirection protocol messages: REDIRECT (redirect flow message, Op Code = 4), RECLAIM (reclaim label message, Op Code = 5), RECLAIM ACK (reclaim label acknowledge message, Op Code = 6), LABEL RANGE (label range message, Op Code = 7), and ERROR (error message, Op Code = 8).

Checksum 386 is the 16-bit one's complement of the one's complement sum of: the source address, destination address and protocol fields from the IP packet encapsulating the IFMP redirection protocol message, and the total length of the IFMP redirection protocol message. Checksum 386 is used by the system for error control purposes.

In IFMP redirection protocol messages, Sender Instance 388 is the sender's instance number for the link, as obtained from the IFMP adjacency protocol. In IFMP redirection protocol messages, Peer Instance field 390 is what the sender believes is the peer's current instance number for the link, as obtained from the IFMP adjacency protocol.

Sequence Number field 392 allows the system node receiving the IFMP redirection protocol message to process IFMP redirection protocol messages in order. Sequence Number 392 is incremented by one, modula 2^{32}, for every IFMP redirection protocol message sent across a link. The IFMP adjacency protocol sets the Sequence Number to zero when the link is reset.

Message Body field 316 contains a list of one or more IFMP redirection protocol message elements. All of the message elements in the list have the same message type because Op Code field 384 applies to the entire IFMP redirection protocol message. The number of message elements included in a single packet must not cause the total size of the IFMP redirection protocol message to exceed the maximum transmission unit (MTU) size of the underlying data link. For Label Range or Error IFMP redirection protocol messages, a single message element is used.

Fig. 9b is a general diagram describing the operation of a system node upon receiving an IFMP redirection protocol message. After startup, the system node receives a packet encapsulating an IFMP redirection protocol message at step 400. In step 402, the system node checks if the IFMP adjacency protocol has currently achieved state synchronization for the link. If state synchronization is not achieved, the system node discards the packet encapsulating the received IFMP redirection protocol message (indicated by step 404). If state synchronization is achieved, the system node in step 406 checks the Source IP Address from the IP header, the Sender Instance 388, and the Peer Instance 390 of the IFMP redirection protocol message packet. If the system node at step 408 determines that the Sender Instance 388 and Source IP Address fields of the incoming IFMP redirection protocol message do not match the values stored by the Update Peer Verifier operation of the IFMP adjacency protocol for the port on which the incoming IFMP redirection protocol message is received, the system node discards the incoming IFMP redirection protocol message packet (step 404). If the values match in step 408, then the system node determines in step 410 whether the Peer Instance field 390 matches the current value for the Sender Instance of the IFMP adjacency protocol. If the values do not match in step 408, then the system node discards the packet (step 404). However, if the values do match in step 408, then the system node continues (step 412) to process the received IFMP redirection protocol message as necessary.

As discussed generally above, an IFMP redirection protocol message may be a REDIRECT message, which is used to instruct an adjacent node to attach one or more labels to packets belonging to one or more specified flows each for a

specified period of time. The system node receiving a REDIRECT message from a downstream node decides whether or not to accept the redirection request made by the REDIRECT message and redirect a flow. The REDIRECT message is not acknowledged in a formal manner. Rather, the actual redirection of packets having attached labels for specified flows indicates the system node's acceptance of the redirection request made by the REDIRECT message. Each REDIRECT message element in the Message Body 394 of the REDIRECT message has the structure shown in Fig. 9c. Described from MSB to LSB, REDIRECT message element 420 includes 8-bit Flow Type field 422, 8-bit Flow ID Length field 424, and 16-bit Lifetime field 426 in a first 32-bit word; 32-bit Label field 428 as the second 32-bit word; and Flow Identifier 430 which is field of integer multiples of 32-bit words. Flow Type field 422 specifies the flow type of the flow identifier contained in Flow Identifier field 430, and Flow ID Length field 424 specifies the length of the Flow Identifier field 430 in integer multiples of 32-bit words. The Lifetime field 426 specifies the length of time (seconds) for which the redirection is valid. As described generally above, after the expiration of the time period specified in the Lifetime field 426, the association of flow identifier and label should be discarded. The Label field 428 contains the label for the specified flow, with the format of the label depending on the type of physical link across which the IFMP redirection protocol message is sent. The Flow Identifier field 430 identifies the flow with which the specified label in the Label field 428 should be associated.

In IFMP redirection protocol message elements, Flow Type 0 has Flow Type = 0 and Flow ID Length = 0; Flow Type 1 has Flow Type = 1 and Flow ID Length = 4; and Flow Type 2 has Flow Type = 2 and Flow ID Length = 3.

The general operation of REDIRECT messages by sender and peer nodes has been discussed in detail above. In addition, other features of the REDIRECT message element include label management and error control. If the label in Label field 428 of REDIRECT message element 420 is outside the range that can be handled across the relevant link, a LABEL RANGE message may be returned to the sender of the REDIRECT message element. The LABEL RANGE message informs the sender of the range of labels that may be sent across the link.

If a system node receives a REDIRECT message element specifying a flow that is already redirected, the system node checks the Label field in the received REDIRECT message element against the label stored for the redirected flow. If the labels match, then the system node resets the lifetime of the redirected flow to that contained in Lifetime field 426 of the received REDIRECT message element. If the labels do not match, then the system node ignores the received REDIRECT message element and the flow is returned to the default state. If the system node detects an error in any of the fields in REDIRECT message element, that particular errored REDIRECT message element is discarded. However, any other error-free REDIRECT message elements that may be in the same IFMP REDIRECT Message Body are not discarded or affected in any way. The system node returns an ERROR message to the adjacent node that sent the errored REDIRECT message element if the system node does not understand the version of the IFMP protocol in the received IFMP message. Also, if the system node does not understand a Flow Type in any of the REDIRECT message elements in the received IFMP message, the system node sends an ERROR message for each Flow Type that is not understood to the adjacent node that sent each particular REDIRECT message element.

As discussed generally above, an IFMP redirection protocol message may be a RECLAIM message, which is used to instruct an adjacent node to unbind or disassociate one or more flows from the labels to which they may be currently bound, and to release the labels for reuse. The system node receiving a RECLAIM message element from a downstream node therefore releases the label and sends to the downstream node a RECLAIM ACK message element as formal acknowledgement of the RECLAIM message. Each RECLAIM message element in the Message Body 394 of a RECLAIM message has the structure shown in Fig. 9d. Described from MSB to LSB, RECLAIM message element 432 includes 8-bit Flow Type field 434, 8-bit Flow ID Length field 436, and 16-bit Reserved field 438 in a first 32-bit word; 32-bit Label field 440 as the second 32-bit word; and Flow Identifier 442 which is field of integer multiples of 32-bit words. Flow Type field 434 specifies the flow type of the flow identifier contained in Flow Identifier field 442, and Flow ID Length field 436 specifies the length of the Flow Identifier field

442 in integer multiples of 32-bit words. In the present embodiment, the Reserved field 438 is unused and set to zero by the system node sending the RECLAIM message element, and is ignored by the system node receiving the RECLAIM message element. The Label field 440 contains the label to be released. The Flow Identifier field 442 identifies the flow with which the specified label in the Label field 440 should be unbound. Each RECLAIM message element applies to a single flow and a single label. After a system node receives a RECLAIM message element, unbinds a flow from a label, returns the flow to the default forwarding state, and releases the label, the system node must issue a RECLAIM ACK message element. RECLAIM ACK message elements may be grouped together into one or more RECLAIM ACK messages and returned to the sender as acknowledgement of the completion of the reclaim operation.

In addition, other features of the RECLAIM message element include label management and error control. If a system node receives a RECLAIM message element specifying an unknown flow, then the system node returns a RECLAIM ACK message element with the same Label 440 and Flow Identifier 442 fields to the sender of the RECLAIM message element. If the system node receives a RECLAIM message element that indicates a known flow but a label in Label field 440 that is not currently bound to that flow, then the system node unbinds that flow and returns that flow to a default forwarding state, as well as issuing a RECLAIM ACK message element containing the actual label to which the flow was previously bound to the sender of the RECLAIM message element. If the system node detects an error in any of the fields in the RECLAIM message element, that particular errored RECLAIM message element is discarded. However, any other error-free RECLAIM message elements that may be in the same IFMP RECLAIM Message Body are not discarded or affected in any way. The system node returns an ERROR message to the adjacent node that sent the errored RECLAIM message element if the system node does not understand the version of the IFMP protocol in the received IFMP message. Also, if the system node does not understand a Flow Type in any of the RECLAIM message elements in the received IFMP message, the system node sends an ERROR message for each Flow Type that is not understood to the adjacent node that sent each particular RECLAIM message element.

As mentioned above, an IFMP redirection protocol message may be a RECLAIM ACK message, which is used to acknowledge the successful release of one or more reclaimed labels. After a system node receiving a RECLAIM message element from a downstream node releases the label, a RECLAIM ACK message element is sent to the node that sent the RECLAIM message element. If possible, each RECLAIM ACK message element should not be sent until all data queued for transmission on the link, using the label specified for release, has been sent. Each RECLAIM ACK message element in the Message Body 394 of a RECLAIM ACK message has the structure shown in Fig. 9e. Described from MSB to LSB, RECLAIM ACK message element 444 includes 8-bit Flow Type field 446, 8-bit Flow ID Length field 448, and 16-bit Reserved field 450 in a first 32-bit word; 32-bit Label field 452 as the second 32-bit word; and Flow Identifier 454 which is field of integer multiples of 32-bit words. Flow Type field 446 specifies the flow type of the flow identifier contained in Flow Identifier field 454, and Flow ID Length field 448 specifies the length of the Flow Identifier field 454 in integer multiples of 32-bit words. In the present embodiment, the Reserved field 450 is unused and set to zero by the system node sending the RECLAIM ACK message element, and is ignored by the system node receiving the RECLAIM ACK message element. The Label field 452 contains the label released from the flow specified by the Flow Identifier field 454. The Flow Identifier field 454 contains the Flow Identifier from the RECLAIM message element that requested release of the label specified in the Label field 452.

Other features of the RECLAIM ACK message element include label management and error control. If a system node receives a RECLAIM ACK message element specifying a flow for which no RECLAIM message element was issued, that RECLAIM ACK message element is ignored. If a system node receives a RECLAIM ACK message element specifying a different label from the label sent in the RECLAIM message for that flow, the system node handles the received RECLAIM ACK message element as if the reclaim operation for the label sent in the RECLAIM message were successful. If the system node detects an error in any of the fields in the RECLAIM ACK message element, that particular errored RECLAIM ACK message element is discarded. However, any other error-free

RECLAIM ACK message elements that may be in the same IFMP RECLAIM ACK Message Body are not discarded or affected in any way. The system node returns an ERROR message to the adjacent node that sent the errored RECLAIM ACK message element if the system node does not understand the version of the IFMP protocol in the received IFMP message. Also, if the system node does not understand a Flow Type in any of the RECLAIM ACK message elements in the received IFMP message, the system node sends an ERROR message for each Flow Type that is not understood to the adjacent node that sent each particular RECLAIM ACK message element.

As discussed above, an IFMP redirection protocol message may be a LABEL RANGE message, which is used in response to a REDIRECT message if the label requested in one or more of the REDIRECT message elements is outside the range that the system node receiving the REDIRECT message can handle. The LABEL RANGE message informs the sender of the REDIRECT message of the label range that can be handled on that link. A single LABEL RANGE message element is used in a LABEL RANGE message. The LABEL RANGE message element in the Message Body 394 of a LABEL RANGE message has the structure shown in Fig. 9f. LABEL RANGE message element 456 includes Minimum Label field 458 as a first 32-bit word, and Maximum Label field 460 as a second 32-bit word. Minimum Label field 458 and Maximum Label field 460, respectively, are the minimum and maximum value of label that can be specified in an IFMP redirection protocol message across a particular link. Only those values of labels within the range from Minimum Label to Maximum Label (inclusive) may be specified in the IFMP redirection protocol message across the link.

As described above, an IFMP redirection protocol message may also be an ERROR message, which may be sent in response to any IFMP redirection protocol message. A single ERROR message element is used in an ERROR message. The ERROR message element in the Message Body 394 of an ERROR message has the structure shown in Fig. 9g. Described from MSB to LSB, ERROR message element 462 includes 8-bit Error Code field 464 and 24-bit Parameter field 466 as a 32-bit word. Error Code field 464 specifies what type of error has occurred. Each ERROR message may specify a single Parameter. If a system node

detects an error in any of the fields in an IFMP redirection protocol message element, that particular errored message element is discarded and an ERROR message is issued. If the system node cannot process or does not understand the particular version of the IFMP protocol in the received IFMP message, the system node sends an ERROR message with Error Code 464 set to the value 1 and with Parameter 466 providing the most recent version of IFMP protocol that the sender is able to understand or process. Also, if the system node does not understand a Flow Type in any of the received IFMP redirect protocol message elements that caused the error, the system node sends an ERROR message with Error Code 464 set to the value 2 and with Parameter 466 providing the Flow Type that caused the error.

2. Flow Labelled Transmission on ATM Data Links

The present invention uses ATM data links to transmit IP packets between system nodes. Packets transmitted on the ATM data links are flow labelled and are encapsulated differently depending on the type of flow, as mentioned above. Using flow classification, the present invention efficiently permits different types of flows to be handled differently (layer 2 routing or layer 3 switching), depending on the type of flow. In addition, each flow type also specifies an encapsulation that is to be used after this type of flow is redirected. In the present embodiment, the system uses encapsulations for ATM data links, as described in detail herein. Of course, encapsulations for each flow type may be specified for different data link technologies for the different hardware switching engines that may be used with the present invention.

A particular flow of packets may be associated with a particular ATM label. Fig. 10a illustrates the format of the 32-bit Label field for ATM labels in the present system. As discussed previously, a label is a virtual path identifier and virtual channel identifier (VPI/VCI), assuming unidirectional virtual channels. Described from MSB to LSB, ATM Label field 470 shown in Fig. 10a includes 4-bit Reserved field 472, 12-bit VPI field 474, and 16 bit VCI field 476. In the present embodiment, Reserved field 472 is set to zero by the sender system node and ignored by the system node receiving the ATM label. For a link that does not

support a full 12-bit VPI, the unused bits in the VPI field 474 are the MSBs in the field 474 and are set to zero. Also, for a link that does not support a full 16-bit VCI, the unused bits in the VCI field 476 are the MSBs in the field 476 and are set to zero.

For any packets in a flow that is not redirected, a system node uses a default encapsulation for the IP packets. If a system node decides that a particular flow type will be redirected, the system node uses an encapsulation particular to each flow type. Upon redirecting a flow, a system node changes the encapsulation used for the redirected flow from the default encapsulation normally used. Rather than using the default encapsulation used for IP packets on the default forwarding channel, the system node uses a different type of encapsulation depending on the flow type that is redirected. It is recognized that an ATM encapsulated IP packet may be an IP packet that is itself encapsulating an IFMP message to and/or from a host computer/server/workstation running a subset of the system software, a basic switching unit, or a switch gateway unit.

As discussed above, in an embodiment of the present invention, three flow types are specified: Flow Type 0, Flow Type 1, and Flow Type 2. Flow Type 0 is used to change the encapsulation of IP packets from the default encapsulation. Flow Type 1 is used for packets carrying data between applications running on stations. Flow Type 2 is used for packets carrying data between stations without identifying the applications that may be running on the stations.

In the present invention, the default encapsulation for IP packets on ATM data links is the Logical Link Control/SubNetwork Attachment Point (LLC/SNAP) encapsulation shown in Fig. 10b. Fig. 10b illustrates a default encapsulated IP packet 480. Basically, the default encapsulation prefixes an LLC/SNAP header to the IP packet which is encapsulated within the payload of an ATM Adaptation Layer type 5 Common Part Convergence Sublayer Protocol Data Unit (AAL-5 CPCS-PDU). Described from MSB to LSB, default encapsulated IP packet 480 includes an LLC/SNAP header (24-bit LLC field 482 followed by an 8-bit portion of SNAP header 484 in the first 32-bit word, and the remaining 32-bit word portion of SNAP header 484), IP packet 486 (which has a length of an integer multiple of 32-bit words), Pad field 488, and AAL-5 CPCS-PDU Trailer field 490.

Pad field 488 may range from 0 to 47 octets, and Trailer field 490 is 8 octets (four 32-bit words). The MOTU of the IP packet 486 using default encapsulation is 1500 octets. The packets using default encapsulation are sent to a predefined VPI/VCI (VPI = 0, VCI = 15, which is the default VPI/VCI according to a specific embodiment (i.e., the packets are forwarded on default virtual channel).

Flow Type 0 encapsulation is used to change the encapsulation of IP packets from the default encapsulation. Fig. 10c illustrates a Flow Type 0 encapsulated IP packet 492. IP packets using Flow Type 0 are encapsulated directly in the payload of an AAL-5 CPCS-PDU without a prefixed LLC/SNAP header. Described from MSB to LSB, Flow Type 0 encapsulated IP packet 492 includes IP packet 494 (which has a length of an integer multiple of 32-bit words), Pad field 496, and AAL-5 CPCS-PDU Trailer field 498. Pad field 496 may range from 0 to 47 octets, and Trailer field 498 is 8 octets (four 32-bit words). The MOTU of the IP packet 494 using Flow Type 0 encapsulation is 1500 octets. The packets belonging to the flow redirected from the default virtual channel use Flow Type 0 encapsulation and are sent to the VPI/VCI specified in the Label field of the IFMP REDIRECT message element encapsulated in IP packet 494 (the IFMP REDIRECT message element encapsulated in IP packet 494 is sent in Flow Type 0 encapsulation).

Default and Flow Type 0 encapsulations do not involve removal of any fields from the IP packet being encapsulated. However, Flow Type 1 and Flow Type 2 "encapsulations" involve removal of certain fields from the IP packet. When these fields are removed, the system node that issued the REDIRECT message stores the removed fields and associates the fields with the ATM virtual channel specified in the ATM label. Accordingly, a complete IP packet may be reconstructed at a destination using the incoming ATM label to access the stored fields.

Flow Type 1 is used for packets carrying data between applications running on stations. Fig. 10d illustrates a Flow Type 1 encapsulated IP packet. IP packets using Flow Type 1 encapsulation are essentially disassembled and selected portions of the disassembled IP packet are encapsulated directly in the payload of an AAL-5 CPCS-PDU, without a prefixed LLC/SNAP header. Described from MSB

to LSB, Flow Type 1 encapsulated IP packet 500 includes 16-bit Total Length field 502 and 16-bit Identification field 504 from the IP header of the disassembled IP packet, as a first 32-bit word. The value of the Total Length field 502 is not changed, but remains the total length of the IP packet before disassembly. Flow Type 1 encapsulated IP packet 500 also includes the 8-bit Flags field 506, 12-bit Fragment Offset field 508, and 16-bit Checksum field 510 from the IP header of the disassembled IP packet, as a second 32-bit word. The transmitted value of Checksum field 510 is the checksum value that would have been computed for the entire IP header if the TTL field had been set to zero. The Version, IHL, TOS, TTL, Protocol, Source Address, and Destination Address fields in the IP header are not transmitted as part of the Flow Type 1 encapsulated IP packet 500. In addition, the first four octets immediately following the IP header (as determined by the IHL) are not transmitted as part of the Flow Type 1 encapsulated IP packet 500. These first four octets correspond to the source port and destination port for TCP and UDP datagrams, as an example. The source port and destination port fields identify the applications running on the stations. Further, Flow Type 1 encapsulated IP packet 500 includes Data 512. Data field 512 is followed by Pad field 514 and AAL-5 CPCS-PDU Trailer field 516. Pad field 514 may range from 0 to 47 octets, and Trailer field 516 is 8 octets (four 32-bit words). The MOTU of the IP packet using Flow Type 1 encapsulation is 1484 octets. The packets belonging to the flow redirected using Flow Type 1 encapsulation are sent to the VPI/VCI specified in the Label field of the corresponding Flow Type 1 IFMP REDIRECT message element encapsulated in the disassembled IP packet (the Label field may be configured to correspond to the source and destination port fields in the TCP or UDP messages).

Flow Type 2 is used for packets carrying data between stations without regard to what applications are running on the stations. Fig. 10e illustrates a Flow Type 2 encapsulated IP packet. IP packets using Flow Type 2 encapsulation are essentially disassembled and selected portions of the disassembled IP packet are encapsulated directly in the payload of an AAL-5 CPCS-PDU, without a prefixed LLC/SNAP header. Described from MSB to LSB, Flow Type 2 encapsulated IP packet 520 includes 16-bit Total Length field 522 and 16-bit Identification field 524 from the IP header of the disassembled IP packet, as a first 32-bit word. The value

of the Total Length field 522 is not changed, but remains the total length of the IP packet before disassembly. Flow Type 2 encapsulated IP packet 520 also includes the 8-bit Flags field 526, 12-bit Fragment Offset field 528, and 16-bit Checksum field 530 from the IP header of the disassembled IP packet, as a second 32-bit word. The transmitted value of Checksum field 530 is the checksum value that would have been computed for the entire IP header if the TTL field had been set to zero. The Version, IHL, TOS, TTL, Protocol, Source Address, and Destination Address fields in the IP header are not transmitted as part of the Flow Type 2 encapsulated IP packet 520. Unlike Flow Type 1 encapsulation, the first four octets immediately following the IP header (as determined by the IHL) are transmitted as part of the Flow Type 2 encapsulated IP packet 520. Further, Flow Type 2 encapsulated IP packet 520 includes Data 532. Data field 532 is followed by Pad field 534 and AAL-5 CPCS-PDU Trailer field 536. Pad field 534 may range from 0 to 47 octets, and Trailer field 536 is 8 octets (four 32-bit words). The MOTU of the IP packet using Flow Type 2 encapsulation is 1488 octets. The packets belonging to the flow redirected using Flow Type 2 encapsulation are sent to the VPI/VCI specified in the Label field of the corresponding Flow Type 2 IFMP REDIRECT message element encapsulated in the disassembled IP packet.

For Flow Type 0, Flow Type 1, and Flow Type 2 encapsulations, the system node that accepts an IFMP REDIRECT message sent by a downstream node stores the removed fields and associates the fields with the ATM virtual channel specified by ATM label to enable cached access information for redirected packets, as discussed above.

B. GSMP

The system software also utilizes the GSMP protocol to establish communication over the ATM link between the switch controller and ATM hardware switching engine of a basic switching unit of the system and thereby enable layer 2 switching when possible and layer 3 IP routing when necessary. In particular, GSMP, a general purpose asymmetric protocol to control an ATM switch, runs on a virtual channel established at initialization across the ATM link between the switch controller and the ATM switch. A single switch controller may

use multiple instantiations of GSMP over separate virtual channels to control multiple ATM switches. GSMP also includes a GSMP adjacency protocol. The GSMP adjacency protocol is used to synchronize state across the ATM link between the switch controller and the ATM switch, to discover the identity of the entity at the other end of the link, and to detect changes in the identity of that entity.

GSMP allows the switch controller to establish and release connections across the ATM switch, add and delete leaves on a point-to-multipoint connection, manage switch ports, request configuration information, and request statistics. GSMP also allows the ATM switch to inform the switch controller of events such as a link going down.

As stated earlier, GSMP is a master-slave protocol. The switch controller issues request messages to the switch. Each request message indicates whether a response from the switch is required and contains a transaction identifier to enable the response to be associated with the particular request. The switch replies with a response message indicating success or failure. In the present embodiment, GSMP has five classes of messages: Connection Management, Port Management, Statistics, Configuration, and Event. Except for the Event message class, the other four classes are request-response message classes, each having a format for the request message and a format for the success response. Unless otherwise indicated, a failure response message is the same as the request message that caused the failure except that the Code field indicates the nature of the failure. In addition to the four request-response message classes, GSMP includes an Event message class, that enables the switch to generate asynchronous Event messages to inform the switch controller of asynchronous events. As Event messages are not acknowledged by the switch controller, Event messages have a single format. In the present embodiment, there are many different message types, i.e., functions of GSMP messages. Each of the five GSMP message classes, except for Port Management, has a number of different message types.

Also, GSMP includes the GSMP adjacency protocol message, which is allocated a particular message type. The GSMP adjacency protocol is used to establish synchronization across the ATM link and maintain a handshake. Except for GSMP adjacency protocol messages, no other GSMP messages may be sent

across the ATM link until the GSMP adjacency protocol has achieved state synchronization. All GSMP messages received on the ATM link that does not currently have state synchronization are discarded.

In the present invention, GSMP packets are variable length and encapsulated directly in an AAL-5 CPCS-PDU with a prefixed LLC/SNAP header, in a similar manner as the default encapsulation for IP packets on ATM data links described above in relation to Fig. 10b. Fig. 11a illustrates an encapsulated GSMP packet 540. Basically, the default encapsulation prefixes an LLC/SNAP header to the GSMP packet which is encapsulated within the payload of an AAL-5 CPCS-PDU. Described from MSB to LSB, default encapsulated GSMP packet 540 includes an LLC/SNAP header (24-bit LLC field 542 followed by an 8-bit portion of SNAP header 544 in the first 32-bit word, and the remaining 32-bit word portion of SNAP header 544), GSMP message 546 (which has a length of an integer multiple of 32-bit words), Pad field 548, and AAL-5 CPCS-PDU Trailer field 550. Pad field 548 may range from 0 to 47 octets, and Trailer field 550 is 8 octets (four 32-bit words). The MOTU of the GSMP message 546 using default encapsulation is 1500 octets. The packets using default encapsulation are sent to the default VPI/VCI, i.e., the default virtual channel.

Fig. 11b illustrates the structure of a GSMP adjacency protocol message 552 that may be contained in GSMP Message field 546 of the encapsulated GSMP packet 540 in Fig. 11a. As seen in Fig. 11b, a GSMP adjacency protocol message 552 includes (described in order of MSB to LSB) the following fields: an 8-bit Version field 554, an 8-bit Message Type field 556, an 8-bit Result field 558, and an 8-bit Code field 560 as a first 32-bit word; Sender Instance field 562 as a second 32-bit word; Sender Port field 564 as a third 32-bit word; Sender Name field 566 as the next 48 bits; Receiver Name field 568 as the next 48 bits; Receiver Port field 570 as the next 32 bits; and Receiver Instance field 572 as the next 32 bits. In discussing GSMP messages, a "sender" is the entity which sends the GSMP message, and a "peer" is the entity to which the sender sends the GSMP message over the ATM link. An entity may be a switch controller or ATM switch.

In a GSMP adjacency protocol message 552, Version field 554 specifies the version of the GSMP protocol which is currently in use (as other

versions may evolve). Message Type field 554 is set to a particular value (Message Type = 96) to specify the GSMP message as being a GSMP adjacency protocol message. Not used for GSMP adjacency protocol messages, Result field 556 is set to zero by a sender entity and ignored by the entity receiving the GSMP adjacency protocol message.

Code field 560 for GSMP adjacency protocol messages specifies the function of the message. In the present embodiment, there are four possible values for Code field 560, i.e., functions of GSMP adjacency protocol messages: SYN (synchronization message, Code = 0), SYNACK (synchronization acknowledge message, Code = 1), RSTACK (reset acknowledge message, Code = 2), and ACK (acknowledge message, Code = 3). In each entity, a timer is required for the periodic generation of SYN, SYNACK, and ACK GSMP messages. For GSMP adjacency protocol purposes, an entity has three possible states for a particular link: SYNSENT (synchronization message sent), SYNRCVD (synchronization message received), ESTAB (synchronization established). State synchronization across a link (when an entity reaches the ESTAB state for a link) is required before the entities may send GSMP messages that are not GSMP adjacency protocol messages. In the present embodiment, the period of the timer is one second, but other periods may be specified. If the timer expires and the sender entity is in the SYNSENT state, the sender entity resets the timer and sends a SYN GSMP adjacency protocol message. If the timer expires and the sender entity is in the SYNRCVD state, the sender entity resets the timer and sends a SYNACK GSMP adjacency protocol message. If the timer expires and the sender entity is in the ESTAB state, the sender entity resets the timer and sends an ACK GSMP adjacency protocol message.

In SYN, SYNACK, and ACK GSMP adjacency protocol messages, Sender Instance 562 is the sender entity's instance number for the link. Indicating a specific instance of a link, an instance number is a 32-bit non-zero number that is guaranteed to be unique within the recent past, and to change when the link comes back after going down or when the identity of the entity at the other end of the link changes. Accordingly, each link has its own unique instance number. Sender Instance 562 is used to detect when a link comes back after going down, or when

the identity of an entity at the other end of the ATM link changes. For a RSTACK GSMP adjacency protocol message, Sender Instance 562 is set to the value of the Receiver Instance field 572 from the incoming GSMP adjacency protocol message that caused the RSTACK message to be generated.

In SYN, SYNACK, and ACK GSMP adjacency protocol messages, Sender Port field 564 is the local port number of the link across which the GSMP message is being sent. As discussed above, port numbers are locally assigned 32-bit values. For RSTACK GSMP adjacency protocol messages, Sender Port field 564 is set to the value of the Receiver Port field 570 from the incoming GSMP adjacency protocol message that caused the RSTACK message to be generated.

For SYN, SYNACK, and ACK GSMP adjacency protocol messages, Sender Name field 566 is the name of the sender entity. The 48-bit Sender Name 566 is unique within the operational context of the basic switching unit. For example, an IEEE 802 MAC address may be used for the Sender Name field. For a RSTACK GSMP adjacency protocol message, the Sender Name field 566 is set to the value of the Receiver Name field 566 from the incoming GSMP adjacency protocol message that caused the RSTACK message to be generated.

For SYN, SYNACK, and ACK GSMP adjacency protocol messages, Receiver Name field 568 is the name of the entity that the sender entity believes is on the other end of the ATM link. If the sender entity does not know the name of that entity, Receiver Name field 568 is set to zero. For a RSTACK GSMP adjacency protocol message, the Receiver Name field 568 is set to the value of the Sender Name field 566 from the incoming GSMP adjacency protocol message that caused the RSTACK message to be generated.

In SYN, SYNACK, and ACK GSMP adjacency protocol messages, Receiver Port field 570 is what the sender entity believes is the local port number for the link that has been allocated by the entity at the other end of the link. If the sender entity does not know the port number of that entity, Receiver Port field 570 is set to zero. For RSTACK GSMP adjacency protocol messages, Receiver Port field 570 is set to the value of the Sender Port field 564 from the incoming GSMP adjacency protocol message that caused the RSTACK message to be generated.

In SYN, SYNACK, and ACK GSMP adjacency protocol messages, Receiver Instance 572 is what the sender entity believes is the current instance number for the link that has been allocated by the entity at the other end of the link. If the sender entity does not know the current instance number at the other end of the link, Receiver Instance field 572 is set to zero. For a RSTACK GSMP adjacency protocol message, Receiver Instance 572 is set to the value of the Sender Instance field 562 from the incoming GSMP adjacency protocol message that caused the RSTACK message to be generated.

Fig. 11c is a simplified diagram illustrating the operation of a sender entity upon receiving an incoming GSMP adjacency protocol message. After startup, the sender entity receives a GSMP adjacency protocol packet at step 582. At step 584, the sender entity determines if the incoming GSMP adjacency protocol message is a RSTACK message. If the incoming GSMP adjacency protocol message is not a RSTACK message (e.g., a SYN, SYNACK, or ACK message), then the sender entity operates in the manner illustrated in the state diagram of Fig. 11d. If the incoming GSMP adjacency protocol message is a RSTACK message, then the sender entity checks at step 584 whether the Sender Instance, Sender Port, and Sender Name fields in the incoming message match the values stored from a previous message by the Update Peer Verifier operation. For the GSMP adjacency protocol, the Update Peer Verifier operation is defined as: storing the values of the Sender Instance, Sender Port, and Sender Name fields from a SYN or SYNACK message received from the entity at the other end of the link. If the values match from step 584, then the sender entity determines at step 586 whether the Receiver Instance, Receiver Port, and Receiver Name fields in the incoming RSTACK message match the values of Sender Instance, Sender Port, and Sender Name fields currently sent in outgoing SYN, SYNACK, and ACK messages for that the port on which the incoming RSTACK message was received. If the values match from step 586, the sender entity determines at step 588 if the sender entity is in the SYNSENT state. If the sender entity is not in the SYNSENT state, the sender entity proceeds to reset the link at step 590. If the values do not match from step 584, or the values do not match from step 586, or the sender entity is in the SYNSENT state, then the sender entity discards the incoming RSTACK message at

step 592 and waits for another packet to arrive. Accordingly when a RSTACK GSMP adjacency protocol message arrives at a sender entity, the sender entity resets the link, as indicated by steps 594, 596, 598, and 600. In step 594, the sender entity generates a new instance number for the link. Then the sender entity in step 596 deletes (i.e., sets to zero) the stored values of Sender Instance, Sender Port, and Sender Name previously stored by the Update Peer Verifier operation. At step 598, the sender entity then sends a SYN GSMP adjacency protocol message, and enters the SYNSENT state in step 600. The sender entity then receives another packet for processing.

Fig. 11d is a state diagram illustrating the operation of a sender entity when the incoming GSMP adjacency protocol message is not a RSTACK message. For the following description of Fig. 11d, condition "%B" is defined as: Sender Instance, Sender Port, and Sender Name fields in the incoming message match the values stored from a previous message by the Update Peer Verifier operation. Condition "%C" in Fig. 11d is defined as: the Receiver Instance, Receiver Port, and Receiver Name in the incoming message match the values of Sender Instance, Sender Port, and Sender Name currently sent in outgoing SYN, SYNACK, and ACK messages. In Fig. 11d, condition "A" signifies that the sender entity receives an incoming SYNACK GSMP adjacency protocol message and that condition %C is met; condition "B" signifies that the sender entity receives an incoming SYNACK GSMP adjacency protocol message and that condition %C is not met; condition "C" signifies that the sender entity receives an incoming ACK GSMP adjacency protocol message and that conditions %B and %C are both met; and condition "D" signifies that the sender entity receives an incoming ACK GSMP adjacency protocol message and that conditions %B and %C are not both met.

If the sender entity is in the SYNSENT state 602 and receives an incoming SYN GSMP adjacency protocol message from a peer on the other end of a link, the sender entity performs an Update Peer Verifier operation and sends a SYNACK GSMP adjacency protocol message to the peer (indicated as step 604). Then the sender goes from SYNSENT state 602 to the SYNRCVD state 606. If the sender receives an incoming SYN GSMP adjacency protocol message while in SYNRCVD state 606, the sender in step 604 performs an Update Peer Verifier

operation and sends a SYNACK GSMP adjacency protocol message to the peer, but remains in the SYNRCVD state 606. If the sender is in SYNRCVD state 606 and either condition B or condition D is met, then the sender sends an RSTACK GSMP adjacency protocol message to the peer (indicated as step 608), and remains in SYNRCVD state 606. If the sender is in the SYNRCVD state 606 and condition C is met, then the sender sends an ACK GSMP adjacency protocol message to the peer (indicated as step 610), and moves to the ESTAB state 612. If the sender is in SYNRCVD state 606 and condition A is met, then the sender performs an Update Peer Identifier operation and sends an ACK GSMP adjacency protocol message to the peer (indicated as step 614), and moves to ESTAB state 612. The sender is and remains in ESTAB state 612, if the sender receives either a SYN or SYNACK GSMP adjacency protocol message or if condition C is met. If condition D is met while the sender is in ESTAB state 612, then the sender remains in ESTAB state 612 and sends a RSTACK GSMP adjacency protocol message (indicated as step 608). While in SYNSENT state 602, if either the sender receives an ACK GSMP adjacency protocol message or condition B is met, then the sender remains in SYNSENT state 602 and sends a RSTACK GSMP adjacency protocol message (step 608). If condition A is met when the sender is in SYNSENT state 602, then the sender performs an Update Peer Verifier operation and sends an ACK GSMP adjacency protocol message (step 614) and enters ESTAB state 612.

In addition to GSMP adjacency protocol messages, other types of GSMP messages 546 include GSMP Connection Management (CM) messages, which are request-response messages. In a basic switching unit, switch controller uses GSMP CM messages to establish, delete, modify, and verify virtual channel connections across the ATM switch. GSMP CM messages may be issued regardless of the status of the switch port, and connections may be established or deleted when a switch port is up, down or otherwise unavailable. Connection Management messages include: Add Branch, Delete Branch, Delete Tree, Verify Tree, Delete All, Move Root, and Move Branch. As mentioned earlier, a virtual channel connection is unidirectional and includes an input virtual channel and at least one output virtual channel or branch. That is, a unicast virtual connection has one output branch, and a multicast virtual connection has two or more output branches.

The Add Branch message is a GSMP CM message used to establish a virtual channel connection or to add an additional branch to an existing virtual channel connection. In present embodiment, no distinction is made between unicast and multicast connections. A first Add Branch message for a particular Input Port, Input VPI, and Input VCI establishes a unicast connection. A second Add Branch message with the same Input Port, Input VPI, and Input VCI establishes converts the unicast connection to a multicast connection by adding another output branch. Other output branches may be added in the same manner with further Add Branch messages. Also, an Add Branch message may be used to check the connection state stored in the ATM switch. The Delete Branch message is a GSMP CM message used to delete a single branch of a virtual channel connection. For example, use of Delete Branch message on a multicast virtual channel connection with two branches removes a branch converting the multicast connection into a unicast connection. The Delete Branch message may also be used to delete a connection by deleting the last branch in a virtual channel connection. Another GSMP CM message, the Delete Tree message is used to delete an entire virtual connection by deleting all remaining branches of the connection. The Verify Tree message is a GSMP CM message used to verify the number of branches on a virtual channel connection. The Delete All message is a GSMP CM message that is used to delete all connections on a switch input port. The Move Root message is a GSMP CM message used to move an entire virtual connection tree from its current Input Port, Input VPI, and Input VCI, to a new Input Port, Input VPI and Input VCI. Another GSMP CM message, the Move Branch message is used to move a single output branch of a virtual channel connection from its current Output Port, Output VPI, and output VCI, to a new Output Port, Output VPI, and Output VCI on the same virtual channel connection.

Fig. 12 illustrates the structure for generic GSMP CM message 620, used as both request and response for Add Branch, Delete Branch, Delete Tree, Verify Tree, and Delete All messages. Generic GSMP CM message 620 may be contained in GSMP Message field 546 of the encapsulated GSMP packet 540 in Fig. 11a. As seen in Fig. 12, generic GSMP CM message 620 includes (described in order of MSB to LSB) the following fields: an 8-bit Version field 622, an 8-bit

Message Type field 624, an 8-bit Result field 626, and an 8-bit Code field 628; 32-bit word Transaction Identifier field 630; 32-bit word Port Session Number field 632; 32-bit word Input Port field 634; 32-bit word Input Label field that includes 4-bit word 636 set to zero, 12-bit Input VPI field 638, and 16-bit Input VCI field 640; 32-bit word Output Port field 642; 32-bit word Output Label field 644 that includes 8-bit word 644 set to zero, 12-bit Output VPI field 646, and 16-bit Output VCI field 648; 16-bit Number of Branches field 650; 8-bit Reserved field 652; and 8-bit Priority field 654.

Except for GSMP adjacency protocol messages, all GSMP messages include the Version field 622, Message Type field 624, Result field 626, Code field 628, and Transaction Identifier field 630, which are used generally in the same manner. For example, Version field 622 in a GSMP message specifies the version of the GSMP protocol which is currently in use (as other versions may evolve). Message Type field 624 is set to a particular value to specify the GSMP message type. For example, a GSMP CM Add Branch message is assigned a specific value for Message Type field 624, and other types of messages are assigned other specific values.

For a GSMP message that is a request message, Result field 626 indicates whether a response is required to the request message when the outcome is successful. Result field 626 in a request message may contain values for NoSuccessAck (indicating no response required if outcome successful) or AckAll (indicating response required if outcome successful). For some types of GSMP request messages, AckAll is the default and a NoSuccessAck value in Result field 626 is ignored. For a GSMP message that is a response message, Result field 626 may contain values for Success (indicating that the request was successful) or Failure (indicating that the request was not successful). A GSMP success response message is not sent until the request has been successfully completed. A GSMP success response message is a copy of the corresponding GSMP request message returned with a Result field 626 indicating Success. For a GSMP request message that does not have a successful outcome, a GSMP failure response message is generated. A GSMP failure response message is a copy of the corresponding GSMP request message returned with Result field 626 indicating Failure. A switch

issuing a GSMP failure response message in response to the failed outcome of a GSMP request message does not modify the connection state within the switch.

In a GSMP response message, Code field 628 provides further information concerning the result. For example, the Code field 628 in a GSMP failure response message may contain an error code specifying the type of error causing the failure. It is recognized that a variety of different codes, failure or other types, may be defined for use in Code field 628. Examples of failure codes that may be defined include: failure specific to the particular message type, unspecified reason not covered by other failure codes, invalid request message, specified request message not implemented on this switch, invalid port session number, at least one specified port does not exist, at least one specified port is down, at least one specified VPI/VCI is out of range on at least one specified port, the specified connection does not exist, the specified output branch does not exist, specified output branch already established for the specified multicast connection on the specified output port, reached the maximum limit of multicast connections supported by switch, reached the maximum limit of branches that the specified multicast connection can support, or general problem relating to multicast capability supported by switch. Of course, other codes may be provided. In addition, Code field 628 may provide further information in a successful response message or an Event message. Code field 628 is not used in GSMP request messages and is set to zero.

Transaction Identifier field 630 is used to associate a GSMP request message with its GSMP response message. In a GSMP request message, the switch controller selects any transaction identifier value for field 630. In a GSMP response message, the value of Transaction Identifier field 630 is set to the value of the transaction identifier from the GSMP request message to which the GSMP response message is responding. Since a GSMP Event message does not require a response, Transaction Identifier field 630 is set to zero.

It is recognized that the above general description of Version, Message Type, Result, Code, and Transaction Identifier fields applies to all GSMP messages, except GSMP adjacency protocol messages. Differences from the general description are described when appropriate.

For GSMP CM messages, Port Session Number field 632 provides the session number of the input port. In particular, the value in Port Session Number field 632 gives the port session number of the switch input port indicated in Input Port field 634. Each switch port maintains a port session number that is assigned by the switch. The port session number remains unchanged while the port is continuously up. However, a new and different port session number is generated after a port is up after being down or unavailable. It is preferred that the new port session number be randomly selected. If the switch controller sends a GSMP CM request message that has an invalid value in Port Session Number field 632, then the switch rejects the GSMP CM request message by sending a GSMP CM failure response message with Code field 628 indicating an invalid port session number causing the failure. A current port session number may be obtained using a GSMP Configuration message.

In a GSMP CM message, Input Port field 634 indicates a switch input port using a 32-bit value assigned by the switch. Input VPI field 638 identifies an ATM virtual path arriving at the switch input port indicated in the Input Port field 634, and Input VCI field 640 identifies an ATM virtual channel arriving on that virtual path identified in Input VPI field 638.

In a GSMP CM message, Output Port field 642 indicates a switch output port using a 32-bit value assigned by the switch. Output VPI field 646 identifies an ATM virtual path departing from the switch output port indicated in the Output Port field 642, and Output VCI field 648 identifies an ATM virtual channel departing on that virtual path identified in Output VPI field 646.

For a GSMP CM message, Number of Branches field 650 gives the number of output branches on a virtual channel connection. Field 650 is used in a GSMP CM Verify Tree message. For all other GSMP CM messages, field 650 is set to zero by the sender entity and ignored by the receiver entity. In the present embodiment, Reserved field 652 which is not used for GSMP CM messages is set to zero by the sender entity and ignored by the receiver entity.

Priority field 654 in a GSMP CM message gives the priority of the connection. The highest priority is numbered zero and the lowest priority is numbered q-1, where q = number of priorities that the switch output port can

support. The q for each switch output port may be obtained from the GSMP Port Configuration message. Each virtual channel connection may be established with a certain quality of service (QOS), by assigning it a priority when it is established. For virtual channel connections that share the same output port, an ATM cell on a connection with a higher priority would be more likely to depart the switch than an ATM cell on a connection with a lower priority, if they are both in the switch at the same time. Priority field 654 is used in GSMP CM Add Branch and Move Branch messages. If a GSMP CM request message (for either Add Branch or Move Branch) has a value in Priority field 654 that the switch does not support, the switch instead assigns the closest priority that it is capable of supporting. In the other GSMP CM messages, Priority field 654 is set to zero by the sender entity and ignored by the receiver entity.

The Add Branch message is a GSMP CM message used to establish a virtual channel connection or to add an additional branch to an existing virtual channel connection. The connection is specified by Input Port field 634, Input VPI field 638, and Input VCI field 640, and the output branch is specified by Output Port field 642, Output VPI field 646, and Output VCI field 648, with the priority of the connection specified by Priority field 654. Also, an Add Branch message may be used to check the connection state stored in the ATM switch. Fig. 13a is a general diagram illustrating the operation of the ATM switch that receives a GSMP Add Branch request message from switch controller. At step 660, switch controller sends a GSMP Add Branch request message that is received by the ATM switch. The ATM switch determines whether the virtual channel connection, as specified in Input Port field 634, Input VPI field 638, and Input VCI field 640 of the received Add Branch request message, exists in the switch, at a step 662. If the switch determines at step 662 that the virtual channel connection does not exist, the ATM switch at step 664 proceeds to establish the connection as specified in the Add Branch request message. If the switch determines at step 662 that the virtual channel connection does exist, then the switch determines at step 666 whether the output branch, as specified by Output Port field 642, Output VPI field 646, and Output VCI field 648 of the received Add Branch request message, exists in the switch. If it is determined that the output branch does not exist, then the ATM

switch proceeds to add a new output branch as specified in the Add Branch request message in step 668. After either steps 664 or 668, the switch determines at step 670 whether the operation was successful. If the operation was not successful, the ATM switch at step 672 sends to the switch controller an Add Branch response message that is a copy of the received Add Branch request message with Result field 626 indicating Failure. The Add Branch response message also may specify the type of failure with the appropriate failure code in its Code field 628. If the operation is determined at step 670 to be successfully completed, the ATM switch at step 674 checks Result field 626 of the Add Branch request message to determine whether a response is required when the request is successful. If the Result field of the request message indicates AckAll, the ATM switch sends a success response to the switch controller in step 676. The Add Branch success response message is a copy of the received Add Branch request message with Result field 626 indicating Success. If the switch determines at step 666 that the output branch specified in the Add Branch request message already exists, then the switch checks in step 680 whether the priority specified in Priority field 654 of the request message is different from the current priority of the output branch. If the switch determines that the requested priority is different than the current priority, the switch changes the priority of the output branch to that specified by the Add Branch request message in step 682. If the priorities are the same, the switch does not change the priority (indicated by 684).

The Delete Branch message is a GSMP CM message used to delete a single branch of a virtual channel connection, or in the case of the last branch to delete the connection. The connection is specified by Input Port field 634, Input VPI field 638, and Input VCI field 640, and the output branch is specified by Output Port field 642, Output VPI field 646, and Output VCI field 648. Fig. 13b is a general diagram illustrating the operation of the ATM switch that receives a GSMP Delete Branch request message from switch controller. At step 690, switch controller sends a GSMP Delete Branch request message that is received by the ATM switch. The ATM switch determines whether the virtual channel connection, as specified in Input Port field 634, Input VPI field 638, and Input VCI field 640 of the received Delete Branch request message, exists in the switch, at a step 692. If

the switch determines at step 692 that the virtual channel connection does exist, the switch at step 694 determines whether the output branch, as specified by Output Port field 642, Output VPI field 646, and Output VCI field 648 of the received Delete Branch request message, exists in the switch. If it is determined that the output branch does exist, then the switch proceeds to delete the output branch as specified in the Delete Branch request message in step 696. After step 696, the switch determines at step 698 whether the deletion operation was successful. If the deletion is determined to be successfully completed, the switch at step 700 determines from Result field 626 of the Delete Branch request message whether a response is required when the request is successful. If the Result field of the request message indicates AckAll (success response required), the switch sends a Delete Branch success response message to the switch controller in step 702. The Delete Branch success response message is a copy of the received Delete Branch request message with Result field 626 indicating Success. If it is determined at step 700 that a success response is not required, then the switch provides no response (indicated as 704). If the switch determines at step 692 that the connection specified in the Delete Branch request message does not exist, or if the switch determines at step 694 that the output branch specified in the Delete Branch request message does not exist, or if the switch determines at step 698 that the deletion operation is unsuccessful, then the switch at step 706 sends a Delete Branch failure response message to the switch controller with the appropriate failure code. A Delete Branch failure response message is a copy of the received Delete Branch request message with Result field 626 indicating Failure and with the type of failure indicated by the appropriate failure code in its Code field 628.

The Delete Tree message is used to delete an entire virtual channel connection by deleting all remaining branches of the connection. The connection is specified by Input Port field 634, Input VPI field 638, and Input VCI field 640. Output Port field 642, Output VPI field 646, and Output VCI field 648 in a Delete Tree message are not used, and are set to zero by the switch controller and ignored by the switch. Fig. 13c is a general diagram illustrating the operation of an ATM switch that receives a GSMP Delete Tree request message from switch controller. At step 710, switch controller sends a GSMP Delete Tree request message that is

received by the ATM switch. The ATM switch determines whether the virtual channel connection, as specified in Input Port field 634, Input VPI field 638, and Input VCI field 640 of the received Delete Tree request message, exists in the switch, at a step 712. If the switch determines at step 712 that the virtual channel connection does exist, then the switch proceeds to delete the connection (and thereby the entire tree) as specified in the Delete Tree request message in step 714. After step 714, the switch determines at step 716 whether the deletion operation was successful. If the deletion is determined to be successfully completed, the switch at step 718 determines from Result field 626 of the Delete Tree request message whether a response is required when the request is successful. If the Result field of the request message indicates AckAll (success response required), the switch sends a Delete Tree success response message to the switch controller in step 720. The Delete Tree success response message is a copy of the received Delete Tree request message with Result field 626 indicating Success. If it is determined at step 716 that a success response is not required, then the switch has no response (indicated as 722). If the switch determines at step 712 that the connection specified in the Delete Tree request message does not exist, or if the switch determines at step 716 that the deletion operation is unsuccessful, then the switch at step 724 sends a Delete Tree failure response message to the switch controller with the appropriate failure code. A Delete Tree failure response message is a copy of the received Delete Tree request message with Result field 626 indicating Failure and with the type of failure indicated by the appropriate failure code in its Code field 628.

The Verify Tree message is a GSMP CM message used to verify the number of branches on a virtual channel connection. The connection is specified by Input Port field 634, Input VPI field 638, and Input VCI field 640. Output Port field 642, Output VPI field 646, and Output VCI field 648 in a Verify Tree message are not used, and are set to zero by the switch controller and ignored by the switch. The number of branches that the switch believes the specified virtual channel connection should contain is given by Number of Branches field 650 in the Verify Tree request message. Fig. 13d is a general diagram illustrating the operation of an ATM switch that receives a GSMP Verify Tree request message from switch controller. At step 730, switch controller sends a GSMP Verify Tree

request message that is received by the ATM switch. The ATM switch determines whether the virtual channel connection, as specified in Input Port field 634, Input VPI field 638, and Input VCI field 640 of the received Verify Tree request message, exists in the switch, at a step 732. If the switch determines at step 732 that the virtual channel connection does exist, then the switch at step 734 checks the actual number of branches for the specified connection and compares the actual number with that in Number of Branches field 650 of the received Verify Tree request message. If the switch determines at step 736 that the numbers match then the verification operation was successful. If the verification is determined to be successfully completed, the switch at step 738 determines from Result field 626 of the Verify Tree request message whether a response is required when the request is successful. If the Result field of the request message indicates AckAll (success response required), the switch sends a Verify Tree success response message to the switch controller in step 740. The Verify Tree success response message is a copy of the received Verify Tree request message with Result field 626 indicating Success. If it is determined at step 738 that a success response is not required, then the switch has no response (indicated as 742). If the switch determines at step 732 that the connection specified in the Verify Tree request message does not exist, then the switch at step 744 sends a Verify Tree failure response message to the switch controller with the appropriate failure code. A Verify Tree failure response message is a copy of the received Verify Tree request message with Result field 626 indicating Failure and with the type of failure indicated by the appropriate failure code in its Code field 628. If the switch determines at step 736 that the verification operation is unsuccessful, then the switch at step 746 sets the actual number of branches into the Number of Branches field 650 of the Verify Tree failure response message and sends it to the switch controller with the Code field 628 set to zero.

The Delete All message is a GSMP CM message that is used to delete all connections on a switch input port. The switch input port is specified by Input Port field 634. In a Delete All message, Input VPI field 638, and Input VCI field 640, Output Port field 642, Output VPI field 646, and Output VCI field 648 are not used, and are set to zero by the switch controller and ignored by the switch. Fig. 13e is a general diagram illustrating the operation of an ATM switch that

receives a GSMP Delete All request message from switch controller. At step 750, switch controller sends a GSMP Delete All request message that is received by the ATM switch. In step 752, the ATM switch determines whether any connections exist on the switch input port specified in Input Port field 634 of the received Delete All request message. If the switch determines at step 752 that connections do exist, then the switch at step 754 proceeds to delete all connections for the switch input port specified in the received Delete All request message. Then the switch determines at step 756 that the complete deletion operation was successful. If the operation is determined to be successfully completed, the switch at step 758 determines from Result field 626 of the Delete All request message whether a response is required when the request is successful. If the Result field of the request message indicates AckAll (success response required), the switch sends a Delete All success response message to the switch controller in step 760. The Delete All success response message is a copy of the received Delete All request message with Result field 626 indicating Success. If it is determined at step 758 that a success response is not required, then the switch has no response (indicated as 762). If the switch determines at step 752 that no connections exist on the switch input port specified in the Delete All request message, then the switch at step 764 sends a Delete All failure response message to the switch controller with the appropriate failure code. A Delete All failure response message is a copy of the received Delete All request message with Result field 626 indicating Failure and with the type of failure indicated by the appropriate failure code in its Code field 628.

The Move Root message is a GSMP CM message used to move an entire virtual connection tree from its current Input Port, Input VPI, and Input VCI, to a new Input Port, Input VPI and Input VCI. Fig. 13f illustrates the structure for GSMP CM Move Root message 770, used as both request and response. GSMP CM Move Root message 770 includes (described in order of MSB to LSB) the following fields: an 8-bit Version field 622, an 8-bit Message Type field 624, an 8-bit Result field 626, and an 8-bit Code field 628; 32-bit word Transaction Identifier field 630; 32-bit word Port Session Number field 632; 32-bit Old Input Port field 772; 4-bit word 774 set to zero, 12-bit Old Input VPI field 776, and 16-bit Old

Input VCI field 778; 32-bit New Input Port field 780; 8-bit word 782 set to zero, 12-bit New Input VPI field 784, and 16-bit New Input VCI field 786; and 32-bit Reserved field 788. Version field 622, Message Type field 624, Result field 626, Code field 628, Transaction Identifier field 630, and Port Session Number 632 are used generally in the same manner as for other GSMP CM messages, as discussed earlier. Reserved field 788 is unused and set to zero by the sender and ignored by the receiver. In a Move Root message, the current virtual channel connection is specified by Old Input Port field 772, Old Input VPI field 776, and Old Input VCI field 778, and the new virtual channel connection is specified by New Input Port field 780, New Input VPI field 784, and New Input VCI field 786. Fig. 13g is a general diagram illustrating the operation of the ATM switch that receives a GSMP Move Root request message from switch controller. At step 790, switch controller sends a GSMP Move Root request message that is received by the switch. The switch determines whether the virtual channel connection, as specified in Old Input Port field 772, Old Input VPI field 776, and Old Input VCI field 778 of the received Move Root request message, exists in the switch, at a step 792. If the switch determines at step 792 that the virtual channel connection does exist, the switch at step 794 determines whether the virtual channel connection, as specified by New Input Port field 780, New Input VPI field 784, and New Input VCI field 786 of the received Move Root request message, is unassigned. If it is determined at step 794 that the virtual channel connection is unassigned, then the switch proceeds to move each output branch of the existing virtual channel connection to establish the new virtual channel connection as specified in the Move Root request message in step 796. After step 796, the switch determines at step 798 whether the move operation was successful. If the operation is determined to be successfully completed, the switch at step 800 determines from Result field 626 of the Move Root request message whether a response is required when the request is successful. If the Result field of the request message indicates AckAll (success response required), the switch sends a Move Root success response message to the switch controller in step 802. The Move Root success response message is a copy of the received Move Root request message with Result field 626 indicating Success. If it is determined at step 800 that a success response is not required, then the switch

provides no response (indicated as 804). If the switch determines at step 792 that the old connection specified in the Move Root request message does not exist, then the switch at step 806 sends a Move Root failure response message to the switch controller with the appropriate failure code. A Move Root failure response message is a copy of the received Move Root request message with Result field 626 indicating Failure and with the type of failure indicated by the appropriate failure code in its Code field 628. If the switch determines at step 794 that the new virtual channel connection specified in the Move Root request message is assigned, then the switch makes no modifications to the existing connections and sets Code field 628 to zero in the Move Root failure response message (indicated at step 808) before sending it to switch controller at step 806.

Another GSMP CM message, the Move Branch message is used to move a single output branch of a virtual channel connection from its current Output Port, Output VPI, and output VCI, to a new Output Port, Output VPI, and Output VCI on the same virtual channel connection. Fig. 13h illustrates the structure for GSMP CM Move Branch message 820, used as both request and response. GSMP CM Move Branch message 820 includes (described in order of MSB to LSB) the following fields: an 8-bit Version field 622, an 8-bit Message Type field 624, an 8-bit Result field 626, and an 8-bit Code field 628; 32-bit word Transaction Identifier field 630; 32-bit word Port Session Number field 632; 32-bit Input Port field 634; 4-bit word 636 set to zero, 12-bit Input VPI field 638, and 16-bit Input VCI field 640; 32-bit Old Output Port field 822; 8-bit word 824 set to zero, 12-bit Old Output VPI field 826, and 16-bit Old Output VCI field 828; 32-bit New Output Port field 830; 8-bit word 832 set to zero, 12-bit New Output VPI field 834, and 16-bit New Output VCI field 836; 24-bit Reserved field 838; and Priority field 940. Version field 622, Message Type field 624, Result field 626, Code field 628, Transaction Identifier field 630, and Port Session Number 632 are used generally in the same manner as for other GSMP CM messages, as discussed earlier. Reserved field 838 is unused and set to zero by the sender and ignored by the receiver. Priority field 940 is used in a similar manner as discussed above for Priority field 654 of a GSMP CM message. In a Move Branch message, the virtual channel connection is specified by Input Port field 634, Input VPI field 638, and Input VCI field 640.

The old branch of the virtual channel connection is specified by Old Output Port field 822, Old Output VPI field 826, and Old Output VCI field 828. The new branch of the virtual channel connection is specified by New Output Port field 830, New Output VPI field 834, and New Output VCI field 836. Fig. 13i is a general diagram illustrating the operation of the ATM switch that receives a GSMP Move Branch request message from switch controller. At step 842, switch controller sends a GSMP Move Branch request message that is received by the switch. The switch determines whether the virtual channel connection, as specified in Input Port field 634, Input VPI field 638, and Input VCI field 640 of the received Move Branch request message, exists in the switch, at a step 844. If the switch determines at step 844 that the virtual channel connection does exist, the switch at step 846 determines whether the old output branch, as specified by Old Output Port field 822, Old Output VPI field 826, and Old Output VCI field 828 of the received Move Branch request message, exists on that virtual channel connection. If it is determined at step 846 that the old output branch exists, then the switch proceeds to add the new output branch as specified by New Output Port field 830, New Output VPI field 834, and New Output VCI field 836 in the Move Branch request message and delete the old output branch as specified in the Move Branch request message in step 848. After step 848, the switch determines at step 850 whether the move operation was successful. If the operation is determined to be successfully completed, the switch at step 852 determines from Result field 626 of the Move Branch request message whether a response is required when the request is successful. If the Result field of the request message indicates AckAll (success response required), the switch sends a Move Branch success response message to the switch controller in step 854. The Move Branch success response message is a copy of the received Move Branch request message with Result field 626 indicating Success. If it is determined at step 852 that a success response is not required, then the switch provides no response (indicated as 856). If the switch determines at step 844 that the virtual channel connection specified in the Move Branch request message does not exist, or if the switch determines at step 846 that the old branch specified in the Move Branch request message does not exist on the virtual channel connection, or if the switch determines at step 850 that the move branch operation

was unsuccessful, then the switch at step 858 does not modify any connection states and sends at step 860 a Move Branch failure response message to the switch controller with the appropriate failure code. A Move Branch failure response message is a copy of the received Move Branch request message with Result field 626 indicating Failure and with the type of failure indicated by the appropriate failure code in its Code field 628.

Providing switch port management, the GSMP Port Management (PM) message allows a port to be brought into service, taken out of service, looped back, or reset. Fig. 14 illustrates the structure for a GSMP PM message 870, used as both request and response messages. GSMP PM message 870 may be contained in GSMP Message field 546 of the encapsulated GSMP packet 540 in Fig. 11a. As seen in Fig. 14, GSMP PM message 870 includes (described in order of MSB to LSB) the following fields: an 8-bit Version field 622, an 8-bit Message Type field 624, an 8-bit Result field 626, and an 8-bit Code field 628; 32-bit word Transaction Identifier field 630; 32-bit word Port field 872; 32-bit word Port Session Number field 874; 32-bit Event Sequence Number field 874; 8-bit Events Flag field 878; 8-bit Duration field 880; and 16-bit Function field 882. Version field 622, Message Type field 624, Result field 626, Code field 628, Transaction Identifier field 630, and Port Session Number 874 are used generally in the same manner as for other GSMP messages, as discussed earlier. Port field 872 gives the port number of the port to which the GSMP PM message applies. A GSMP PM message has a particular Message Type field and various possible functions that may be specified in Function field 882. Some of the functions of GSMP PM messages include: a Bring Up function, a Take Down function, an Internal Loopback function, an External Loopback function, a Bothway Loopback function, a Reset Input Port function, and a Reset Event Flags function. Each switch port maintains an Event Sequence Number and a set of Event Flags (one Event Flag for each type of Event Message). The Event Sequence Number is set to zero when the port is initialized and is incremented each time an asynchronous event reportable by an Event message is detected on that port, regardless of whether the Event message is sent or not. When a switch port sends an Event message, it sets the corresponding Event Flag on that port. The port is not permitted to send another Event message of the

same type until the corresponding Event Flag is reset by a Reset Event Flags function of a GSMP PM message. The use of the Event Flags provides simple flow control to prevent the switch from flooding the switch controller with Event messages. In a GSMP PM request message, Event Sequence Number field 876 is not used and is set to zero by the switch controller and ignored by the switch. In a GSMP PM success response message, Event Sequence Number field 876 gives the current value of the Event Sequence Number of the switch port specified in the received GSMP PM request message. In a GSMP PM request message with the Function field 882 specifying Reset Event Flags, particular bits in the Event Flags field 878 may be used to reset the corresponding Event Flags in the switch port specified by the Port field 872. In a GSMP PM success response message with the Function field 882 specifying Reset Event Flags, the bits in Event Flags field 878 are set to the current values of the corresponding Event Flags for the specified port, after the Event Flags specified in the request message have been reset. By setting the Event Flags field to all zeros in a GSMP PM message with a Reset Event Flags function, the switch controller is able to obtain the current state of the Event Flags and the current Event Sequence Number of the specified port without changing the state of the Event Flags. In other GSMP PM messages with a different Function field 882 specified, the Event Flags field 878 is not used and is set to zero by the switch controller and ignored by the switch. Duration field 880 is used only in GSMP PM messages with the Function field 882 specified as Internal Loopback, External Loopback, or Bothway Loopback. Duration field 880 provides the length of time (in seconds) that any of the loopback states remains in operation. When the duration expires, the port which was in loopback automatically returns to service. In GSMP PM messages with a different Function field 882 specified, Duration field 880 is not used and is set to zero by the switch controller and ignored by the switch. In GSMP PM messages, Function field 882 specifies the action to be taken (the specified action is taken regardless of the current status of the port). The Bring Up function brings the port into service, and the Take Down function takes the port out of service. The Internal Loopback function performs an internal loopback (ATM cells arriving at the output port from the switch fabric are looped through to the input port back to the switch fabric). The External Loopback function performs

an external loopback (ATM cells arriving at the input port from the external communications link are looped back to the communications link at the physical layer without entering the input port). The Bothway Loopback function performs both internal and external loopback. The Reset Input Port function resets the input port (all connections arriving at the specified input port are deleted and the input and output port hardware are reinitialized so that all VPI/VCI values for the specified input port in the connection table are empty). The Reset Event Flags function resets the Event Flags as discussed above.

GSMP Statistics messages allow the switch controller to request values of various hardware counters associated with the switch input and output ports, and virtual channels. Two classes of statistics messages are provided: VC Activity messages, and Port and VC Statistics messages. The VC Activity message is used to determine whether one or more specific virtual channels have recently been carrying traffic. A VC Activity message contains one or more VC Activity Records. Each VC Activity Record is used to request and return activity information concerning a single specified virtual connection. If a switch supports traffic accounting per virtual connection, the current value of the traffic counter for each specified virtual connection is returned in the VC Traffic Count field of the VC Activity Record. The current value of the traffic count is compared to previous values for each of the specified virtual connections to determine whether each virtual connection has been active in the intervening period. If a switch supports traffic detecting per virtual connection in some other way besides traffic accounting, the result may be indicated for the virtual connection using a Flag field in the VC Activity Record. The Port and VC Statistics messages are used to query the various port and VC specific traffic and error counters. A Port Statistics message is used to get the statistics of the switch port specified in the Port field of the message, and a VC Statistics message is used to get the statistics for the virtual channel (specified in the VPI/VCI fields of the message) on the switch input port specified in the Port field of the message.

GSMP Configuration messages permit the switch controller to determine the capabilities of the ATM switch in basic switching unit. Three message types for GSMP Configuration messages are defined: Switch

Configuration, Port Configuration, and All Ports Configuration. GSMP Configuration messages use different formats for the request message and the response message, since they contain different information in their fields. Sent by switch controller to an ATM switch, a Switch Configuration request message, indicated by a particular Message Type field, asks the ATM switch for its global configuration. Then the switch returns to the switch controller a Switch Configuration response message that includes fields for the switch type and switch name of the ATM switch, as well as the version of the switch control firmware installed. The switch type is a allocated by a manufacturer of the switch to identify the switch product, and the switch name may be a 48-bit IEEE 802 MAC address or other quantity that is unique within the operational context of the switch. A Port Configuration request message has its own particular Message Type field and is sent by switch controller to an ATM switch. The Port Configuration request message asks the switch for configuration information of a single switch port that is specified in the Port field of a Port Configuration request message. The switch sends to the switch controller a Port Configuration success response message that includes configuration information for both the input and output sides of the specified port. The configuration information in a Port Configuration success response message includes: the current Port Session Number of the port, the minimum value of VPI that the connection table on the input port that can be supported by GSMP, the maximum value of VPI that the connection table on the input port that can be supported by GSMP, the minimum value of VCI that the connection table on the input port that can be supported by GSMP, and the maximum value of VCI that the connection table on the input port that can be supported by GSMP. The configuration information also includes: the cell rate (rate of ATM cells per second) of the port, the current status (i.e., down, up, unavailable, internal loopback, external loopback, or bothway loopback) of the port; the port type (the type of physical transmission interface of the port, e.g., unknown, SONET STS-3c at 155.52 Mbps, DS3 at 44.736 Mbps, 4B/5B encoding at 100 Mbps, 8B/10B encoding at 155.52 Mbps, 25 Mbps ATM Forum physical layer, or 51 Mbps ATM Forum physical layer); and the number of priorities that the output port can assign to virtual channel connections. The configuration information provided is referred

to as the Port Record for a port. The switch controller sends an All Ports Configuration request message, which has its own particular Message Type field, to the ATM switch to ask for the configuration information for all of the switch ports. Thus, the All Ports Configuration request message does not specify a particular port. The switch sends an All Ports Configuration success response message that provides: the number of Port Records contained by the response message, the byte length of each Port Record, and the Port Records for each port. The Port Record for each port is the same configuration information discussed for the Port Configuration success response message. Of course, if the number of Port Records exceeds a specified maximum amount set for the All Ports Configuration success response message, then the Port Records may be sent in multiple success response messages that each do not exceed the specified maximum amount.

GSMP Event messages allow the ATM switch to inform the switch controller of certain asynchronous events. As mentioned earlier, Event messages are not acknowledged. Event messages may have different Message Types, depending on the asynchronous event. Different Event messages include a Port Up Event message, a Port Down Event message, an Invalid VPI/VCI Event message, a New Port Event message, and a Dead Port Event message. Each switch port maintains an Event Sequence Number and a set of Event Flags (one Event Flag for each type of Event Message). When a switch port sends an Event message, it sets the corresponding Event Flag on that port. The port is not permitted to send another Event message of the same type until the corresponding Event Flag is reset by a Reset Event Flags function of a GSMP Port Management message. The use of the Event Flags provides simple flow control to prevent the switch from flooding the switch controller with Event messages. The Event Sequence Number is set to zero when the port is initialized and is incremented each time an asynchronous event reportable by an Event message is detected on that port, regardless of whether the Event message is sent or not. The current Event Sequence Number is included in Event messages to inform the switch controller of asynchronous events that have occurred on the port, but that have not been reported via an Event message due to the action of the simple flow control mechanism. A Port Up Event message informs the switch controller that the specified port has changed from the down

state to the up state. When a port comes up, all connections on its input port are deleted (the input port's connection tables are empty) and a new Port Session Number is assigned by the switch. A Port Down Event message informs the switch controller that the specified port has changed from the up state to the down state. If a switch is capable of detecting link failure, the switch sends a Port Down Event message to report link failure to the switch controller. When one or more ATM cells arrive at an input port with a VPI/VCI that is not currently allocated to a virtual channel connection, the switch sends an Invalid VPI/VCI Event message to the switch controller. The Invalid VPI/VCI Event message specifies the input port and the VPI/VCI in the Port and VPI/VCI fields respectively. A New Port Event message specifying the number of a new port informs the switch controller that the new port has been added to the switch. The Dead Port Event message informs the switch controller that a port has been removed from the switch. The Dead Port Event message specifies the number of the removed port and the Port Session Number that was valid before the port was removed in its Port and Port Session Number fields respectively.

C. IFMP-C

According to some embodiments, the system software also may utilize the IFMP-C protocol to establish communication over the link between a basic switching unit's switch controller (referred herein as the IFMP-C controller) and a switching agent (referred herein as the IFMP-C agent) and thereby distribute layer 3 packet forwarding when desired. In particular, IFMP-C, a general purpose asymmetric protocol to control an IFMP-C agent, runs on a virtual channel established at initialization across the link between the IFMP-C controller and the IFMP-C agent. A single IFMP-C controller may use multiple instantiations of IFMP-C over separate virtual channels to control multiple IFMP-C agents. IFMP-C also includes an IFMP-C adjacency protocol, which is used to synchronize state across the link between the IFMP-C controller and the IFMP-C agent, to discover the identity of the entity at the other end of the link, and to detect changes in the identity of that entity.

IFMP-C allows the IFMP-C controller to establish and release connections across the ATM switch, add and delete leaves on a point-to-multipoint connection, manage IFMP-C interfaces, request interface configuration information, and request statistics. IFMP-C also allows the IFMP-C agent to inform the IFMP-C controller of events such as a link going down.

As stated earlier, IFMP-C is a master-slave protocol. The IFMP-C controller issues request messages to the IFMP-C agent. Each request message indicates whether a response from the IFMP-C agent is required and contains a Transaction Identifier to enable the response to be associated with the particular request. The IFMP-C agent replies with a response message indicating success or failure.

IFMP-C includes an IFMP-C adjacency protocol, which is allocated a set of particular message types. The IFMP-C adjacency protocol is used to establish synchronization across the link and to maintain a handshake between the IFMP-C controller and the IFMP-C agent. Except for IFMP-C adjacency protocol messages, no other IFMP-C messages may be sent across the link until the IFMP-C adjacency protocol has achieved state synchronization. All other IFMP-C messages received on the link that does not currently have state synchronization are discarded. In the present embodiment, IFMP-C has four other classes of messages: Interface, Branch, Control, and Management. These four classes are request-response message classes, each having a format for the request message and a format for the success response message. Unless otherwise indicated, a failure response message is the same as the request message that caused the failure except that the Code field indicates the nature of the failure. In the present embodiment, there are many different message types, i.e., functions of IFMP-C messages. Each of the IFMP-C message classes has a number of different message types, in a specific embodiment.

In the present invention, IFMP-C packets are variable length and encapsulated directly in an AAL-5 CPCS-PDU with a prefixed LLC/SNAP header, in a similar manner as the default encapsulation for IP packets on ATM data links described above in relation to Fig. 10b. Fig. 15a illustrates an encapsulated IFMP-C packet 1000. Basically, the default encapsulation prefixes an LLC/SNAP header to the IFMP-C packet which is encapsulated within the payload of an AAL-5 CPCS-

PDU. Described from MSB (transmitted first) to LSB, default encapsulated IFMP-C packet 1000 includes an LLC/SNAP header (24-bit LLC field 1002 followed by an 8-bit portion of SNAP header 1004 in the first 32-bit word, and the remaining 32-bit word portion of SNAP header 1004), IFMP-C message 1006 (which has a length of an integer multiple of 32-bit words), Pad field 1008, and AAL-5 CPCS-PDU Trailer field 1010. Pad field 1008 may range from 0 to 47 octets, and Trailer field 550 is 8 octets (four 32-bit words). The MOTU of the IFMP-C message 1006 using default encapsulation is 1500 octets. The packets using default encapsulation are sent to VPI = 0, VCI = 15 (default virtual channel), in a specific embodiment.

Fig. 15b illustrates the generic structure of a typical IFMP-C message 1012 that may be contained in IFMP-C Message field 1006 of the encapsulated IFMP-C packet 1000 in Fig. 15a. As seen in Fig. 15b, the generic IFMP-C message 1012 includes (described in order of MSB to LSB) the following fields: an 8-bit Version field 1014, an 8-bit Message Type field 1016, an 8-bit Code field 1018, and an 8-bit Flags field 1020 as a first 32-bit word; a 16-bit Transaction Identifier field 1022, and a 16-bit Message Length field 1024 as a second 32-bit word; a Senders Sync Instance field 1026 as a third 32-bit word; a Peer Sync Instance field 1028 as a fourth 32-bit word; and a Message Body field 1030. In discussing IFMP-C messages, a "sender" is the entity which sends the IFMP-C message, and a "peer" is the entity to which the sender sends the IFMP-C message over the link. An entity may be an IFMP-C controller or IFMP-C agent. The generic IFMP-C message 1012 discussed above has some differences (discussed below) in relation to IFMP-C adjacency protocol messages.

When sending a message, the sender sets the Version field 1014 to correspond to the version number of the IFMP-C protocol currently being used (as other IFMP-C protocol versions evolve). Message Type field 1016 identifies the type and format of Message Body field 1030. Each particular message has a unique message type in the specific embodiment. Code field 1018 is used to indicate the success or failure of an operation, and Flag field 1020 is used to indicate how the packet should be handled and what kind of response is required. Flag field 1020 may be set as an acknowledge-complete (PLEASE_ACK) flag when the sender requires notification from the receiver that the operation has completed. When the

sender requires notification only if the operation failed, the sender may set Flag field 1020 as an acknowledge-negative (PLEASE_NACK). If the operation is successful and the request message had a PLEASE_ACK flag in Flag field 1020, Flag field 1020 should be set as an acknowledge (ACK) flag and Code field 1018 should be set to a predetermined value, such as 0 in the specific embodiment, in the response message. However, if the operation failed and the request message had a PLEASE_NACK flag in Flag field 1020, Flag field 1020 of the response message should be set as a negative-acknowledgement (NACK) flag and Code field 1018 should indicate the specific cause of the failure. A complete list of failures for use in Code field 1018 is discussed later. Transaction Identifier field 1022 uniquely identifies each message. When a message requires a response, the response message has the same value in Transaction Identifier field 1022, thereby allowing messages to be correlated. Message Length field 1024 gives the length (including the IFMP-C header but not including any SNAP/LLC encapsulation) of the message in octets, according to a specific embodiment. Senders Sync Instance field 1026 contains the value identifying the sender's current synchronization instance. This value is exchanged during the IFMP-C adjacency protocol, as described below. Upon receiving a message from a sender, the value in Senders Sync Instance field 1026 is compared with the value of the local peer instance at the receiver from the IFMP-C adjacency protocol. If the instances do not match, then the packet is ignored by the receiver. Peer Sync Instance field 1028 contains the value identifying what the sender believes to be the current synchronization instance of its peer. This value is exchanged during the IFMP-C adjacency protocol. Upon receiving a message from the sender, the value in Peer Sync Instance filed 1028 is compared with the local instance at the receiver from the IFMP-C adjacency protocol. If the instances do not match, then the receiver ignores the packet. Message Body field 1030 contains any message-specific data, as described in more detail below for the various message class types.

Fig. 16a illustrates the generic structure of an IFMP-C adjacency protocol message 1040 that may be contained in IFMP-C Message field 1006 of the encapsulated IFMP-C packet 1000 in Fig. 15a. As seen in Fig. 16a, IFMP-C adjacency protocol message 1040 includes (described in order of MSB to LSB):

Version field 1014; Message Type field 1016; Code field 1018; Flags field 1020; Transaction Identifier field 1022; Message Length field 1024; a 32-bit Sender Instance field 1042; a 32-bit Peer Instance field 1044; a 16-bit Sender Type field 1046 and a 16-bit ACK Interval field 1048; a Sender Name field 1050 as the next 48 bits; and a Peer Name field 1052 as the next 48 bits. In discussing IFMP-C adjacency protocol messages, a "sender" is the entity which sends the IFMP-C adjacency protocol message, and a "peer" is the entity to which the sender sends the IFMP-C adjacency protocol message over the link. An entity may be an IFMP-C controller or an IFMP-C agent.

In IFMP-C adjacency protocol message 1040, Version field 1014, Message Type field 1016, Flags field 1020, Transaction Identifier field 1022, and Message Length field 1024 are used as discussed above for generic IFMP-C message 1012 in Fib. 15b. Message Type field 1016 of IFMP-C adjacency protocol message 1040 is set to a particular value to specify the particular type of IFMP-C adjacency protocol message. In the present embodiment, there are four possible values for Message Type field 1016 in IFMP-C adjacency protocol messages: SYN (synchronization message), SYNACK (synchronization acknowledge message), RSTACK (reset acknowledge message), and ACK (acknowledge message). Sender Instance field 1042 is set to the sender's current instance number for the link. The instance number is used to detect when one side of the link has restarted. The instance number for a link should be unique in the recent past and should be unique when a node restarts. Peer Instance field 1044 is set to what the sender believes is the current instance number of the remote side of the link. A predetermined value, such as 0 in a specific embodiment, is used in Peer Instance field 1044 to indicate when the sender does not know the remote instance. This predetermined value may be reserved for this purpose and is not be used as a valid instance number for use in Sender Instance field 1042. Sender Type field 1046 indicates the type of IFMP-C entity (defined types are IFMP-C controller and IFMP-C agent) sending the message. When an entity receives any IFMP-C adjacency message, the receiver compares the Sender Type field 1046 in the received message with the receiver's type in order to prevent an IFMP-C controller from forming adjacency with another IFMP-C controller or an IFMP-C agent from forming adjacency with another

IFMP-C agent. If the types are the same or undefined, then the message is ignored. If the types are complementary (one is an IFMP-C controller and one is an IFMP-C agent), then the synchronization proceeds. ACK Interval field 1048 is set by the IFMP-C controller to indicate its preferred acknowledgement (ACK) interval. The IFMP-C agent uses the ACK interval as the timeout interval for running the adjacency protocol. When sending IFMP-C adjacency protocol messages, the IFMP-C agent sets ACK Interval field 1048 to a predetermined value, such as 0 in a specific embodiment, which is ignored by the IFMP-C controller. Sender Name field 1050 is a value unique within the operational context of the device (such as a Media Access Controller (MAC) address if available) that identifies the sender. Peer Name field 1052 is set by the sender to what it believes is the name of the peer on the remote side of the link. The sender sets Peer Name field 1052 to a predetermined value, such as 0 in a specific embodiment, to indicate when the peer name is unknown.

As discussed above, the IFMP-C adjacency protocol is used to establish state synchronization across a link connecting an IFMP-C controller and an IFMP-C agent, as well as identifying link state changes and the other side of the link restarting. Each side of the link runs the IFMP-C adjacency protocol. For IFMP-C adjacency protocol purposes, three possible states for a particular link exist: SYNSENT (synchronization message sent), SYNRCVD (synchronization message received), ESTAB (synchronization established). State synchronization across a link (when an adjacency is established, the interfaces will be in the ESTAB state) is required before the IFMP-C controller and the IFMP-C agent may send IFMP-C messages.

In the IFMP-C adjacency protocol, there are two types of events that can cause state changes: timer-driven events and packet arrivals. These state changes are illustrated in Fig. 16b, which is a state diagram illustrating the operation of a sender entity (either an IFMP-C controller or an IFMP-agent) in the three possible states of the IFMP-C adjacency protocol. The IFMP-C controller sets the timer interval on the IFMP-C agent by setting ACK Interval field 1048 to a specific value, such as 1 second in a specific embodiment. Of course, other values for the timer interval may be used in other embodiments. In each entity, a timer is

required for the periodic generation of SYN, SYNACK, and ACK IFMP-C adjacency protocol messages, as discussed below.

Timer-driven events for the IFMP-C adjacency protocol are discussed herein. As shown in Fig. 16b, if the timer expires (indicated by t) and the sender entity is in the SYNSENT state 1060, the sender entity resets the timer and sends a SYN IFMP-C adjacency protocol message (indicated by 1062). This action (indicated by dotted line) is performed only by the IFMP-C agent, who has the responsibility of initiating synchronization by sending the SYN packet, according to a specific embodiment. If the timer expires and the sender entity is an IFMP-C controller, the entity merely resets the timer without sending a SYN packet. If the timer expires and the sender entity is in the SYNRCVD state 1064, the sender entity resets the timer and sends a SYNACK IFMP-C adjacency protocol message (indicated by 1066). If the timer expires and the sender entity is in the ESTAB state 1068, the sender entity resets the timer and sends an ACK IFMP-C adjacency protocol message (indicated by 1070). To time out the IFMP-C adjacency protocol, both sides of the links should reset the link and enter the SYNSENT state if they go through a predetermined number (for example, three) of timeout periods without receiving an ACK packet from the other side of the link. When the IFMP-C agent times out, it should restore the ACK Interval to its default value.

In addition to timer-driven transitions discussed above, state transitions in the IFMP-C protocol also are caused by IFMP-C adjacency packet arrival, as discussed herein. When an IFMP-C adjacency message arrives at an entity, an action is taken based on the current state of the entity, the message contents, and the message type. The following operations are performed during the IFMP-C adjacency protocol: an Update Peer operation and a Reset Link operation. Each instance of the IFMP-C adjacency protocol keeps state that defines the peer instance and the peer name (i.e. the instance and name of the peer entity on the remote side of the link). The Update Peer operation sets the local peer state to match the sender state from the most recently received IFMP-C adjacency message. Taken when restarting the synchronization, the Reset Link operation generates a new instance number for the local side of the link and deletes all peer state by setting the peer instance and peer name to zero.

For the following description in Fig. 16b, condition "%X" is defined as: Peer Instance and Peer Name fields in the incoming message match the local values of instance and name associated with the link. Condition "%Y" in Fig. 16b is defined as: the Sender Instance and Sender Name fields in the incoming message match the values of Sender Instance and Sender Name stored for the peer instance and peer name. In Fig. 16b, condition "A" signifies that the sender entity receives an incoming SYNACK IFMP-C adjacency protocol message and that condition %X is met; condition "B" signifies that the sender entity receives an incoming SYNACK IFMP-C adjacency protocol message and that condition %X is not met; condition "C" signifies that the sender entity receives an incoming ACK IFMP-C adjacency protocol message and that conditions %X and %Y are both met; condition "D" signifies that the sender entity receives an incoming ACK IFMP-C adjacency protocol message and that conditions %X and %Y are not both met; condition "E" signifies that the sender entity receives an incoming RSTACK IFMP-C adjacency protocol message and that conditions %X and %Y are both met; condition "F" signifies that the sender entity receives an incoming RSTACK IFMP-C adjacency protocol message and that conditions %X and %Y are not both met; condition "G" signifies that the sender entity receives either a SYN or SYNACK IFMP-C adjacency protocol message and that condition %X is met; and condition "H" signifies that the sender entity receives either a SYN or SYNACK IFMP-C adjacency protocol message and that condition %X is not met.

Many possible transitions are possible when an entity is in SYNSENT state 1060, as described herein. As seen in Fig. 16b, if the sender entity is in the SYNSENT state 1060 and receives an incoming SYN IFMP-C adjacency protocol message from a peer on the other end of a link, the sender entity performs an Update Peer operation and sends a SYNACK IFMP-C adjacency protocol message to the peer (indicated as step 1072). Then the sender goes from SYNSENT state 1060 to the SYNRCVD state 1064. If the sender receives an incoming RSTACK IFMP-C adjacency protocol message while in SYNSENT state 1060, the sender entity remains in SYNSENT state 1060 but drops the message (indicated by 1074). If the sender either receives an incoming ACK IFMP-C adjacency protocol message or condition B is met, then the sender entity remains in SYNSENT state 1060 but

sends an RSTACK IFMP-C adjacency protocol message (indicated by 1076). If condition A is met when in SYNSENT state 1060, the sender entity performs an Update Peer operation and sends an ACK IFMP-C adjacency protocol message (indicated by 1078) and transitions to ESTAB state 1068.

When an entity is in SYNRCVD state 1064, many possible transitions are possible, as described herein. As seen in Fig. 16b, if the sender entity is in the SYNRCVD state 1064 and receives an incoming SYN IFMP-C adjacency protocol message from a peer on the other end of a link, the sender entity performs an Update Peer operation and sends a SYNACK IFMP-C adjacency protocol message to the peer (indicated as step 1072) while remaining in SYNRCVD state 1064. If condition B or D is met while in SYNRCVD state 1064, the sender entity remains in SYNRCVD state 1064 and sends an RSTACK IFMP-C adjacency protocol message (indicated by 1078). If condition F is met while the entity is in SYNRCVD state 1064, the entity remains in SYNRCVD state 1064 but drops the message (indicated by 1080). If condition E is met while in SYNRCVD state 1064, the sender entity performs a Reset Link operation and sends a SYN IFMP-C adjacency protocol message (indicated by 1082), and transitions from SYNRCVD state 1064 to SYNSENT state 1060. If condition A is met while in SYNRCVD state 1064, the sender entity performs an Update Peer operation and sends an ACK IFMP-C adjacency protocol message (indicated by 1078), and transitions from SYNRCVD state 1064 to ESTAB state 1068. If condition C is met while in SYNRCVD state 1064, the sender entity sends an ACK IFMP-C adjacency protocol message (indicated by 1084), and transitions from SYNRCVD state 1064 to ESTAB state 1068.

When an entity is in ESTAB state 1068, many possible transitions are possible, as described herein. As seen in Fig. 16b, if condition D or H is met while in ESTAB state 1068, the sender entity remains in ESTAB state 1068 and sends an RSTACK IFMP-C adjacency protocol message (indicated by 1088). If condition F is met while the entity is in ESTAB state 1068, the entity remains in ESTAB state 1068 but drops the message (indicated by 1080). If condition E is met while in ESTAB state 1068, the sender entity performs a Reset Link operation and sends a SYN IFMP-C adjacency protocol message (indicated by 1086), and

transitions from ESTAB state 1068 to SYNSENT state 1060. If condition C or G is met while in ESTAB state 1068, the sender entity sends an ACK IFMP-C adjacency protocol message (indicated by 1090) and remains in ESTAB state 1068. Each entity on a side of a link should not send more than one ACK IFMP-C adjacency protocol message generated by a packet arrival per timeout period, according to a specific embodiment.

An IFMP-C Interface message may be contained in IFMP-C Message field 1006 of the encapsulated IFMP-C packet 1000 in Fig. 15a. As mentioned above, after synchronization of a link is established, IFMP-C Interface messages are used to discover and configure the interfaces on the IFMP-C agent. In a specific embodiment, IFMP-C Interface messages include Interface List request and response messages, an Interface List errors message, Interface Query request and response messages, an Interface Query errors message, Interface Configuration request and response messages, and an Interface Configuration Errors message. IFMP-C Interface List messages and Interface Query messages have specific values in Message Type field 1016. Of course, additional messages are possible or messages performing similar functions are possible in other embodiments.

Interface List request and response messages and Interface List errors messages are used to determine what interfaces are available on the IFMP-C agent. Figs. 17a and 17b illustrate the structure of Interface List request and response messages, respectively. As seen in Fig. 17a, an Interface List request message 1100 has the generic format as previously described in relation to Fig. 15b, with Message Body field 1030 containing a Next Cookie field 1112, which is a 32-bit value returned from a previous interface list response. Next Cookie field 1112 is used to support interface lists that span more than one message. The value in Next Cookie field 1112 is a value returned by the IFMP-C agent to allow the next message request to continue at the place in the interface list the previous message terminated. A predefined value, such as 0 in a specific embodiment, is used to indicate that the interface list should start from the beginning. If the Interface List request message has the PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, the IFMP-C agent returns an Interface List response message 1114 (seen in Fig. 17b) with its Flags field 1020 set to an ACK flag and

Code field 1018 set to the predetermined value (0 in the specific embodiment) indicating no error. As shown in Fig. 17b, Next Cookie field 1112 in Interface List response message 1114 is a 32-bit value returned as part of the message. If the value in Next Cookie field 1112 is 0, then all of the interfaces have been enumerated. If the value is not 0, then that value is used as an argument to the next interface list message to get any remaining interfaces. The IFMP-C agent assigns each interface a unique 32-bit identifier, which is used in other IFMP-C messages to refer to a specific interface. Interface List response message 1114 lists in the Interface Identifier 1 field 1118, Interface Identifier 2 field 1120, etc. all the identifiers for each interface on the IFMP-C agent that can be listed in the response message.

If the Interface List request failed, the IFMP-C agent returns an Interface List errors message that consists of the IFMP-C header. In the Interface List errors message, the NACK flag should be set in Flags field 1020, Code field 1018 should indicate the cause of the failure, and Transaction Identifier field 1022 should be the same as in the Interface List request message. Exemplary causes of failure, each having a specific value for use in Code field 1018, include that the cookie was invalid, the IFMP-C agent was unable to allocate a message buffer to complete the response, a client-specific error prevented the request from completing, or other causes.

Interface Query request and response messages and Interface Query errors messages are used to obtain the attributes associated with a specified interface on the IFMP-C agent. Figs. 17c and 17d illustrate the structure of Interface Query request and response messages, respectively. As seen in Fig. 17c, an Interface Query request message 1130 has the generic format as previously described in relation to Fig. 15b with Message Body field 1030 containing an Interface Identifier field 1132, which is a 32-bit identifier that uniquely identifies the interface whose attributes are being queried. As described above, interface identifiers are assigned by the IFMP-C agent and are obtained through the Interface List messages. If Interface Query request message 1130 from IFMP-C controller has the PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, the IFMP-C agent returns Interface Query response message 1134 (shown in Fig. 17d)

having an ACK flag set in its Flags field 1020 and Code field 1018 set to the predetermined value indicating no error. As seen in Fig. 17d, the Message Body field 1030 of Interface Query response message 1134 also includes a 48-bit Interface Name field 1136 for the name of the queried interface (the MAC address can be used in a specific embodiment), an 8-bit Interface Type field 1138 (values may be defined for various interfaces types such as ATM, Ethernet, FastEthernet, Gigabit Ethernet, FDDI, or other LAN interfaces), an 8-bit Media Type field 1140 (indicating the interface's physical media, such as multimode fiber, Category 5 twisted pair, single mode fiber, etc.). Additionally, the Interface Query response message 1134 includes a 32-bit Supported Speeds field 1142 and a 32-bit Current Speed field 1144. Supported Speeds field 1142 indicates the different transmission rates (e.g., 10 Megabits per second (Mbps), 25 Mbps, 100 Mbps, 155 Mbps, 622 Mbps, 1000 Mbps, as well as others) that the queried interface supports. For interfaces supporting more than one transmission rate, several flags can be set in field 1142. An auto negotiation flag may be set in Supported Speeds field 1142 if the interface supports auto negotiation of the speed settings. Current Speed field 1144 indicates the current transmission rate of the queried interface. If the interface is in auto-configure mode, the current interface speed is indicated in Current Speed field 1144 and the auto negotiation flag is set in Supported Speeds field 1142. IFMP-C Interface Query response message 1134 also includes a 32-bit Supported Duplex field 1146 (indicating duplex rates supported by the queried interface, such as half duplex, full duplex, or auto negotiation of duplex setting; more than one flag may be set for interfaces supporting more than one duplex setting), a 32-bit Current Duplex field 1148 (indicating the current duplex setting of the interface; if the interface is in auto negotiation mode for duplex setting, field 1146 will so indicate). Further, Interface Query response message 1134 includes a 32-bit Interface Slot Identifier field 1150 (identifies the physical slot that the interface occupies on the IFMP-C agent), a 32-bit Interface Port Identifier field 1152 (identifies the physical port that the interface occupies on the IFMP-C agent), a 16-bit Interface Flags field 1154, and a 16-bit Interface Status field 1156. Interface Flags field 1154 specifies the current settings of configuration options on the queried interface, with each flag indicating a different state (such as the interface being administratively up, being in

a promiscuous mode, accepting all multicast packets, etc.). Interface Status field 1156 indicates current status information (such as IFMP-C control traffic is running over this interface, or the link on this interface is up, etc.) about the link that is not mutable by the IFMP-C controller. Still further, Interface Query response message 1134 includes a 32-bit Minimum Receive Label field 1158, a 32-bit Maximum Receive Label field 1160, a 32-bit Minimum Transmit Label field 1162, and a 32-bit Maximum Transmit Label field 1164. If the interface is ATM, Minimum Receive Label field 1158 and Maximum Receive Label field 1160 respectively indicate the minimum and maximum VCI on which the interface can receive. If the interface is ATM, Minimum Transmit Label field 1162 and Maximum Transmit Label field 1164 respectively indicate the minimum and maximum VCI on which the interface can transmit. If the queried interface is not ATM, fields 1158, 1160, 1162 and 1164 are set to zero.

If Interface Query request fails, then the IFMP-C agent sends an Interface Query errors message that consists of an IFMP-C header having a NACK flag set in its Flags field 1020, the appropriate cause of failure indicated in its Code field 1018, and its Transaction Identifier field 1022 the same as in the Interface Query request message 1130. Exemplary causes of failure that may be indicated in Code field 1018 of the Interface Query errors message include the interface identifier being invalid, the IFMP-C agent being unable to allocate a message buffer to complete the response, a client-specific error preventing the request from completing, as well as others.

Interface Configuration request and response messages and Interface Configuration errors messages are used to allow the IFMP-C controller to change the interface configuration on the IFMP-C agent. Fig. 17e illustrates the structure of an Interface Configuration request message 1170. As seen in Fig. 17a, an Interface Configuration request message 1170 has the generic format as previously described in relation to Fig. 15b with Message Body field 1030 containing an Interface Identifier field 1132 (uniquely identifying the interface whose configuration is to be changed), a 16-bit Clear Flags field 1172, a 16-bit Set Flags field 1174, a 32-bit Speed field 1176, and a 32-bit Duplex field 1178. Clear Flags field 1172 indicates which flags that the IFMP-C agent should clear on the specified

interface to affect the operation of the interface. Examples of clearing flags include taking the interface administratively down, taking the interface out of promiscuous mode, preventing the interface from receiving all multicast packets, etc. Set Flags field 1174 indicates which flags that the IFMP-C agent should set on the specified interface to affect the operation of the interface. Examples of setting flags include making the interface administratively up, putting the interface into promiscuous mode, allowing the interface to receive all multicast packets, etc. Speed field 1176 is used to allow the IFMP-C controller to change the speed setting (among the interface's supported speeds, such as auto negotiation, 10 Mbps, 25 Mbps, 100 Mbps, 155 Mbps, 622 Mbps, 1000 Mbps, or others) of the specified interface. Duplex field 1178 allows the IFMP-C controller to change the duplex setting (among the interface's supported duplex settings, such as auto negotiated duplex, half duplex, or full duplex) of the specified interface. Setting Speed field 1176 or Duplex field 1178 to a predefined value such as 0 in a specific embodiment indicates that the speed or duplex setting should not be changed. If Interface Configuration request message 1170 from IFMP-C controller has the PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, the IFMP-C agent returns an Interface Configuration response message consisting of the IFMP-C header fields (all the fields shown in Fig. 15b except the Message Body field) having an ACK flag set in its Flags field 1020 and Code field 1018 set to the predetermined value indicating no error.

If the Interface Configuration request message failed, the IFMP-C agent returns an Interface Configuration errors message that consists of the IFMP-C header. In the Interface List errors message, the NACK flag should be set in Flags field 1020, Code field 1018 should indicate the cause of the failure, and Transaction Identifier field 1022 should be the same as in the Interface List request message. Exemplary causes of failure, each having a specific value for use in Code field 1018, include an invalid interface identifier, invalid configuration parameters are used, a client-specific error prevented the request from completing, or other causes.

In addition to IFMP-C Adjacency and Interface messages, other types of IFMP-C messages 1012 include IFMP-C Branch request, response, and errors

messages. Six types of IFMP-C Branch messages exist to create, modify and delete forwarding branches. Specifically, IFMP-C Branch messages include: Add Branch, Delete Branch, Delete Tree, Move Branch, Read Branch, and Get Tree Statistics. Each forwarding branch is given a preference value. When multiple matches occur, the branch with the lowest preference is used.

As mentioned earlier, the basic forwarding is done through branches, where each branch consists of two components: the input data and the output data. The input data provides enough information to allow incoming packets to be matched to branches, and the output data is the information necessary to forward the packet that matches the input data. The basic operations used to modify the IFMP-C agent's forwarding state are Add Branch and Delete Branch. Adding more than one branch with the same input data results in a Tree, which can be used to forward each incoming packet to multiple destinations. For example, a first Add Branch message having a particular input data and some output data establishes unicast packet forwarding. A second Add Branch message with the same input data and other output data converts the unicast forwarding to a multicast forwarding by adding another output branch associated with the same input data. Other output branches may be added in the same manner with further Add Branch messages. The Delete Branch message is an IFMP-C Branch message used to delete a single branch. Use of a Delete Branch message on a multicast connection with two branches removes a branch having specified output data, converting the multicast forwarding into a unicast forwarding. Another IFMP-C Branch message, the Delete Tree message is used to delete all branches sharing the same input data. Another IFMP-C Branch message, the Move Branch message is used to change existing output information on an existing branch. Additional IFMP-C Branch messages include Get Tree Statistics messages, which are used to obtain statistics of the forwarding entry whose input data is specified, and Read Branch messages, which are used for diagnostic and debugging purposes to allow the IFMP-C controller to retrieve all forwarding branches on an IFMP-C agent.

The Add Branch message and the Delete Branch request messages are IFMP-C Branch messages that use the same message format 1200 (but have different message types), shown in Fig. 18a. As seen in Fig. 18a, IFMP-C

Add/Delete Branch request message format 1200 has the generic format as previously described in relation to Fig. 15b with Message Body field 1030 that includes: Input Interface Identifier field 1201, a 16-bit Input Precedence field 1202, a 16-bit Input Flags field 1204, a 32-bit Output Interface Identifier field 1206, a 24-bit Reserved field 1208, an 8-bit Key Length field 1210, a 8-bit Output Header Length field 1212, a 8-bit Remove Length field 1214, a 8-bit Transformation Type field 1216, a 8-bit Transformation Data Length field 1218, a 32-bit Quality of Service Handle field 1222, a predefined length Input Key Data field 1224, a predefined length Input Key Mask field 1226, a predefined length Output Header Data field 1228, and a Transformation Data field 1230.

The fields shown in Fig. 18a (other than the generic IFMP-C header fields shown in Fig. 15b) are described herein. Assigned by the IFMP-C agent and obtained through IFMP-C Interface List messages, Input Interface Identifier field 1201 uniquely identifies the specific input interface to which the input branch is to be applied. Input Precedence field 1202, a 16-bit unsigned integer in a specific embodiment, denotes the precedence assigned to the branch. When matching incoming packets to input keys, the key with the lowest precedence is matched first. If more than one entry has the same precedence, then the IFMP-C agent may select any of the matching branches to forward the packet. Input Flags field 1204 applied to the input branch and flags indicated in this field are used to denote specific behavior that should be taken if packets match this forwarding entry. Examples of such behavior that may be flagged include: "fall through" - searching at the next precedence level after a packet is transmitted, rather than terminating; or "drop" - dropping all packets that match this input entry. Output Interface Identifier field 1206 uniquely identifies the interface to use for transmitting the packet. Reserved field 1208, which may be reserved for future use, may be set by a sender entity to 0 and ignored by the receiver entity in a specific embodiment if this field is not used. Key Length field 1210, a 8-bit unsigned integer in a specific embodiment, gives the length of the Input Key Data field 1224 and the Input Key Mask field 1226 in octets. Output Header Length field 1212, a 8-bit unsigned integer in the specific embodiment, gives the length of the Output Header Data field 1228 in octets. Remove Length field 1214, a 8-bit unsigned integer, specifies a number of octets to

remove from the beginning of the packet before applying the transformation indicated in Transformation Data field 1230.

Transformation Type field 1216 specifies the type of modification (for example, no modification, IP packet to IFMP flow type 1, IP packet to IFMP flow type 2, IFMP flow type 1 to IP packet, IFMP flow type 2 to IP packet, standard IP forwarding, truncating packet, or other modifications) that is to be made to the packet before it is transmitted. Some of the types of modifications require data that is not part of the packet, so this required data is provided in Transformation Data field 1230. Specifically, "IP packet to IFMP flow type 1 (or 2)" transformation type converts from IPv4 (or other IP packet version in use) to the encapsulation for IFMP flow type 1 (or 2) previously discussed. "IFMP flow type 1 (or 2) to IPv4 packet" transformation type converts a packet arriving as an IFMP flow type 1 (or 2) to a IPv4 packet using Transformation Data field 1230 to reconstruct the removed fields as well as updating the TTL and IP header checksum. "Standard IP forwarding" transformation type decrements the TTL in the IP header and updates the IP header checksum. The "Truncate packet" transformation type limits the size of the packet to that specified (in Transformation Data field 1230) by the forwarding data.

As shown in Fig. 18a, Transformation Data Length field 1218 is a 8-bit unsigned integer specifying the length of the Transformation Data field 1230 that is included in the Add/Delete Branch message format 1200. If the transformation does not require additional data in Transformation Data field 1230, then Transformation Data Length field 1218 is set to zero. Quality of Service Handle field 1222 indicates how packets matching the branch should be handled by the IFMP-C agent. Its length specified by the Key Length field 1210, Input Key Data field 1224 contains the data that is compared with the incoming packet to see if it matches the forwarding entry. The data in Key Data field 1224 holds link level information such as MAC address, layer 3 information such as IP address, etc. Its length specified by the Key Length field 1210, Key Mask field 1226 is used to specify the relevant bits of the key data, when comparing the information in Input Key Data field 1224 to the incoming packet. Its length specified by Output Header Length field 1212, Output Header Data field 1228 contains a header (typically link

level information such as a MAC header on an Ethernet interface, or the VPI/VCI on ATM interfaces) that should be prepended to the packet before it is transmitted. As mentioned above, Transformation Data field 1230 contains the data required to perform the specified transformation. For "IFMP flow type 1 (or 2) to IPv4" transformation, the data in Transformation Data field 1230 is similar to the flow identifiers shown in Fig. 7a (or Fig. 7b for Flow type 2, except that Type of Service and Protocol fields may be Reserved in some embodiments). When "Truncate packet" transformation is specified so that a partial copy of the packet is sent to a specific destination, the data 1240 in Transformation Data field 1230 includes a 16-bit Reserved field 1242 that may be set to zero by the sender and ignored by the receiving entity, and a 16-bit Truncate Length field 1244, as shown in Fig. 18b. Truncate Length field 1244 denotes how many bytes of the packet should be transmitted out the interface. If the arriving packet is longer than this number, it is truncated to this length. This truncated length is the number of bytes after any bytes have been removed and does not include any output header that may be added.

If an IFMP-C Add (or Delete) Branch request message has the PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, the IFMP-C agent returns an IFMP-C Add (or Delete) Branch response message 1250 with the format shown in Fig. 18c, having an ACK flag set in its Flags field 1020 and Code field 1018 set to the predefined value indicating no error. As shown in Fig. 18c, the response message has IFMP-C header fields of the generic IFMP-C message, with the message body containing a 16-bit Reserved field 1252 and a 16-bit Output Count field 1254. Reserved field 1252 is reserved for future use and may be set to zero by the sender and ignored by the receiver. Output Count field 1254, an unsigned integer, contains the number of output branches that share the same input information after the current operation is applied (if branch count is 1, then it is a unicast branch; if branch count is greater than 1, then it is a multicast branch). This field 1254 is used by the IFMP-C controller to help verify the consistency of the branch state after each operation.

If the IFMP-C Add Branch (or Delete Branch) request failed, then the IFMP-C agent sends an IFMP-C Add Branch (or Delete Branch) errors message.

The errors message consists of the IFMP-C header, which the same Transaction Identifier field 1022 as the request message, the NACK flag set in its Flags field 1020 and its Code field 1018 set to indicate the cause for the failure. Exemplary causes for failure include, for example: if one of the interface identifiers is invalid; if the input key length is longer than the maximum supported by the IFMP-C agent; the output transformation is unsupported or unrecognized by the IFMP-C agent; the IFMP-C agent has insufficient resources to complete the request; another branch exists with the same input key but has different flags; the quality of service parameters are invalid or unsupported by the IFMP-C agent; the input key or mask is not supported by the IFMP-C agent; the specified branch attempting to be added already exists on the IFMP-C agent; the specified branch attempting to be deleted does not exist on the IFMP-C agent; a client-specific error prevented the completion of the request; or other causes.

As shown in Fig 18d, the IFMP-C Delete Tree request message 1260 has a message format containing many of the fields described for Fig. 18a. Delete Tree request message format 1260 has the generic format as previously described in relation to Fig. 15b with Message Body field 1030 that includes: Input Interface Identifier field 1201, Input Precedence field 1202, Input Flags field 1204, a 56-bit Reserved field 1262, Key Length field 1210, predefined length Input Key Data field 1224, and predefined length Input Key Mask field 1226.

Input Interface Identifier field 1201 uniquely identifies the input interface to which the input branch (for which the Tree is to be deleted) should be applied. Input Key Data field 1224 provides the data that is compared with the incoming packet to see if it matches the forwarding entry to be deleted, and Input Key Mask field 1226 specifies the relevant bits of the key data when comparing the input key data to the incoming packet. Reserved field 1262 is reserved for future use and may be set to zero by the sender and ignored by the receiver.

If IFMP-C Delete Tree request message 1260 has a PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, then the IFMP-C agent returns an IFMP-C Delete Tree response message which is the IFMP-C header fields (as shown in Fig. 15b except without the Message Body field), with the ACK flag set in its Flags field 1020 and its Code field 1018 set to the

predefined value indicating no error. If IFMP-C Delete Tree request fails, then the IFMP-C agent returns an IFMP-C Delete Tree errors message which is the IFMP-C header fields (as shown in Fig. 15b except without the Message Body field), with the NACK flag set in its Flags field 1020 and its Code field 1018 set to identify the cause of the failure (such as invalid interface identifier, no branch exists with input key as specified in the request, another branch exists with same input key as specified but with different flags, or other causes).

Fig. 18e illustrates the structure of an IFMP-C Move Branch request message 1300, containing many of the fields described for Fig. 18a. Move Branch request message format 1300 has the generic format as previously described in relation to Fig. 15b with Message Body field 1030 that includes (described MSB to LSB): Input Interface Identifier field 1201, Input Precedence field 1202, Input Flags field 1204, a 32-bit Old Output Interface Identifier field 1302, a 24-bit Reserved field 1304, 8-bit Key Length field 1210, a 8-bit Old Output Header Length field 1308, a 8-bit Old Remove Length field 1310, a 8-bit Old Transformation Type field 1312, a 8-bit Old Transformation Data Length field 1314, a 32-bit Old Quality of Service Handle field 1318, a 32-bit New Output Interface Identifier field 1320, a 32-bit Reserved field 1322, a 8-bit New Output Header Length field 1324, a 8-bit New Remove Length field 1326, a 8-bit New Transformation Type field 1328, a 8-bit New Transformation Data Length field 1330, a 32-bit New Quality of Service Handle field 1334, predefined length Input Key Data field 1224, predefined length Input Key Mask field 1226, a predefined length Old Output Header Data field 1340, a predefined length Old Transformation Data field 1342, a predefined length New Output Header Data field 1344, and a predefined length New Transformation Data field 1346. Many of the fields listed above are described above for Figs. 18a, and various other fields in Move Branch request message 1300 are easily understood by simply noting that the values of the fields defining the output data of the old branch are replaced with the values in the fields defining the output data of the new branch taking the old branch's place.

If IFMP-C Move Branch request message 1300 has a PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, then the IFMP-C agent returns an IFMP-C Move Branch response message which has the same

format 1250 as the IFMP-C Add/Delete Branch response message (seen in Fig. 18c) with the ACK flag set in its Flags field 1020 and its Code field 1018 set to the predefined value indicating no error. If IFMP-C Move Branch request fails, then the IFMP-C agent returns an IFMP-C Move Branch errors message which is the IFMP-C header fields (as shown in Fig. 15b except without the Message Body field), with the NACK flag set in its Flags field 1020 and its Code field 1018 set to identify the specific cause of the failure (such as invalid interface identifier, input key length longer than maximum length supported, insufficient resources at IFMP-C agent, output transformation unsupported or unrecognized, no branch exists with same input key as specified for the original branch in the request, the original branch does not exist and no branch matches the new branch, another branch exists with same input key as specified but with different flags, quality of service parameters invalid or unsupported, a client-specific error prevents the completion of the request, or other causes).

Being yet another IFMP-C Branch message type, IFMP-C Get Tree Statistics messages are used by the IFMP-C controller to determine when entries are no longer being used so that these entries may be reclaimed. The IFMP-C agent keeps a running counter of each time an entry is used to forward a packet. This entry is kept for each tree, as each branch on the tree will be used the same number of times. As seen in Fig. 19a, an IFMP-C Get Tree Statistics request message 1400 has the generic format as previously described in relation to Fig. 15b with Message Body field 1030 that includes a list of tree data information: Tree Data 1 field 1402, and Tree Data 2 field 1404 in a specific embodiment. Additional Tree Data fields also may be included in the list contained as Message Body field 1030. Fig. 19b illustrates the Tree Data field structure 1406, which all Tree Data fields use. Each Tree Data field structure 1406 includes (MSB to LSB) Input Interface Identifier field 1201, Input Precedence field 1202, Input Flags field 1204, a 40-bit Reserved field 1408, Key Length field 1210, a 18-bit Record Size field 1410, a 64-bit Usage Count field 1412, predefined length Input Key Data field 1224, and predefined length Input Key Mask field 1226. Input Interface Identifier field 1201 uniquely identifies the input interface to which the input branch (for which the Tree Statistics are to be obtained) should be applied. Reserved field 1408 is reserved for future

use and may be set to zero by the sender and ignored by the receiver. Record Size field 1410, which indicates the size of the particular tree record (for example, in Tree Data 1 field 1402), is used to find the beginning of the next tree record (for example, in Tree Data 2 field 1404). Usage Count field 1412 is a 64-bit unsigned integer which is incremented each time the IFMP-C agent uses the specified tree to forward a packet. In a request message, Usage Count field 1412 is set to zero by the sender and is ignored by the receiver. The remaining fields are not described herein as they have already been described in relation to Fig. 18a.

If IFMP-C Get Tree Statistics request message 1400 has a PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, then the IFMP-C agent returns an IFMP-C Get Tree Statistics response message, which has the same format as the request message 1400 (Fig. 19a), with the ACK flag set in its Flags field 1020 and its Code field 1018 set to the predefined value indicating no error. The Get Tree Statistics response message also returns the appropriate counter values in the Usage Count fields of the Tree Data fields.

If IFMP-C Get Tree Statistics request fails, then the IFMP-C agent returns an IFMP-C Get Tree Statistics errors message which is the IFMP-C header fields (as shown in Fig. 15b except without the Message Body field) of the request message, with the NACK flag set in its Flags field 1020 and its Code field 1018 set to identify the specific cause of the failure (such as invalid interface identifier, one of the specified output trees does not exist on the IFMP-C agent, another tree exists with same input key as specified but with different flags, or other causes).

Another of the IFMP-C Branch message types are IFMP-C Read Branch messages which are used for diagnostic and debugging purposes to allow the IFMP-C controller to retrieve all forwarding branches on an IFMP-C agent. To enumerate all of the branches on the IFMP-C agent, the Read Branch message uses a "getnext" operation. Every time an entry is ready, the IFMP-C agent returns the branch information as well as a cookie to use as an argument to the next read operation. The cookie is opaque to the IFMP-C controller and is used by the IFMP-C agent to remember where it left off on the last read. A predefined value, such as 0 in a specific embodiment, may be reserved and used by the IFMP-C controller to get the initial entry on the IFMP-C agent. A successive string of get

forwarding entry requests will be successful until all of the tables are enumerated at which point the IFMP-C agent returns an indication that the end of the list has been reached.

Figs. 20a and 20b illustrate the structure of IFMP-C Read Branch request message 1420 and IFMP-C Read Branch response messages 1430, respectively. As seen in Fig. 20a, IFMP-C Read Branch request message 1420 has the generic format as previously described in relation to Fig. 15b, with Message Body field 1030 containing Input Interface Identifier field 1201 and a 32-bit Next Cookie field 1422. Input Interface Identifier field 1201 uniquely identifies the input interface on which the branches should be read. Next Cookie field 1422 is an opaque 32-bit value returned as part of the previous Read Branch response message. The value in Next Cookie field 1112 is a state kept by the IFMP-C agent to keep track of the location of the last Read Branch request. A predefined value, such as 0 in a specific embodiment, is used to indicate that starting from the beginning of the list should be done.

If the IFMP-C Read Branch request message has the PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, the IFMP-C agent returns an IFMP-C Read Branch response message 1430 (seen in Fig. 20b) with its Flags field 1020 set to an ACK flag and Code field 1018 set to the predetermined value (0 in the specific embodiment) indicating no error. As shown in Fig. 20b, Read Branch response message 1430 has the generic format as previously described in relation to Fig. 15b, with Message Body field 1030 containing: Input Interface Identifier field 1201, Input Precedence field 1202, Input Flags field 1204, Output Interface Identifier field 1206, a 24-bit Reserved field 1432, Key Length field 1210, Output Header Length field 1212, Remove Length field 1214, Transformation Type field 1216, Transformation Data Length field 1218, Quality of Service Handle field 1222, Next Cookie field 1422, predefined length Input Key Data field 1224, predefined length Input Key Mask field 1226, predefined length Output Header Data field 1228, and Transformation Data field 1230. Reserved field 1432 is reserved for future use and is set to zero by the sender and ignored by the receiver. Next Cookie field 1422 is the state kept by the IFMP-C agent to keep track of the location of the last request and is used as the input to the next Read Branch

message. The remaining fields are not described herein, as they have been described earlier in relation to Fig. 18a.

If the Read Branch request failed, the IFMP-C agent returns a Read Branch errors message that consists of the IFMP-C header. The Read Branch errors message should be identical to the IFMP-C header of the Read Branch request message except with the NACK flag set in Flags field 1020 and with Code field 1018 set to indicate the cause of the failure. Exemplary causes of failure include that one of the interface identifiers in the message was invalid, the cookie was invalid, the IFMP-C agent was unable to allocate a message buffer to complete the response, a client-specific error prevented the request from completing, or other causes.

In addition to IFMP-C Adjacency, Interface and Branch messages, IFMP-C messages include an IFMP-C Control messages, such as an IFMP-C Reset message. The IFMP-C Reset message is used to re-initialize the state of the IFMP-C agent when the IFMP-C controller has lost communication with the IFMP-C agent or if the IFMP-C controller believes that IFMP-C agent is corrupted. The IFMP-C Reset request message instructs the IFMP-C agent to reset all of its state to the initial condition. Upon receiving the Reset request message, the IFMP-C agent removes all forwarding branches and initializes the interfaces without resetting the IFMP-C adjacency protocol state. The IFMP-C Reset message consists of the IFMP-C header shown in Fig. 15b (without the Message Body field) with Message Type field 1016 set to denote the message as a reset message. If the Reset request message has a PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, then the IFMP-C agent returns a Reset response message that consists of the identical IFMP-C header as the Reset request message, except with its Flags field 1020 set as an ACK flag and its code field set to a predefined value indicating no error. If the Reset request fails, then the IFMP-C agent returns a Reset errors message consisting of the identical IFMP-C header as the Reset request message, except with its Flags field 1020 set as a NACK flag and its code field set to a value indicating the specific cause for the error (such as a client-specific error preventing completion of the request).

In addition to IFMP-C Adjacency, Interface, Branch, and Control messages, the IFMP-C protocol also includes IFMP-C Management messages, which are used to get the information needed for network management and diagnostic purposes. IFMP-C Management messages include various message types: IFMP-C Node Information messages and IFMP-C Interface Statistics messages.

IFMP-C Node Information messages obtain information (such as software version number, etc.) about the node running IFMP-C. An IFMP-C Node Information request message has the generic format of IFMP-C Header (without Message Body field 1030, as shown in Fig. 15b) with Message Type field 1016 identifying the message as an IFMP-C Node Information message.

If the IFMP-C Node Information request message has the PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, the IFMP-C agent returns an IFMP-C Node Information response message 1440 (seen in Fig. 21a) with its Flags field 1020 set to an ACK flag and Code field 1018 set to the predetermined value (0 in the specific embodiment) indicating no error. As shown in Fig. 21a, IFMP-C Node Information response message 1440 has the generic format as previously described in relation to Fig. 15b, with Message Body field 1030 containing: a 48-bit Node ID field 1442, a 48-bit Parent ID field 1444, a 16-bit Node Type field 1446, a 15-bit Reserved field 1448, a 32-bit Parent Slot field 1450, a 32-bit Parent Shelf field 1452, a 16-bit Firm Minor Version field 1454, and a 16-bit Firm Major Version field 1456. Node ID field 1442 is a value (such as a MAC address for the node) that uniquely identifies the node. For situations where the node is part of a larger system such as a board in a chassis, Parent ID field 1444 is set to a value (such as a MAC address for the parent) that uniquely identifies the container (or parent) of the node. If the node is not part of a larger system such as a board in a chassis, Parent ID field 1444 is set to a predefined value, such as 0 in a specific embodiment, indicating no parent node. Node Type field 1446 is a 16-bit unsigned integer which indicates the type of the IFMP-C node. The value in Node Type field 1446 may be assigned by the vendor so that the combination of the Organization Unique Identifier (OUI, which are the high-order 24 bits of a MAC address) portion of the Node ID and the Node Type are unique

for each type of IFMP-C agent. Reserved field 1448 is reserved for future use, and may be set to 0 by the sender and ignored by the receiver. If the node is part of a larger system such as a board in a chassis, then Parent Slot field 1450 is set to correspond to the slot that the node occupies in the parent container. If the IFMP-C agent can not determine the slot information or is not part of a larger container, then Parent Slot field 1450 may be set to 0. If the node is part of a larger system such as a board in a chassis, then Parent Shelf field 1452 is set to correspond to the slot that the node occupies in the parent container. If the IFMP-C agent can not determine the shelf information or is not part of a larger container, then Parent Shelf field 1452 may be set to 0. Firm Minor Version field 1454 indicates the minor version of the firmware of the IFMP-C agent currently running, and Firm Major Version field 1456 identifies the major version of the firmware of the IFMP-C agent currently running.

If the Node Information request failed, then the IFMP-C agent returns a Node Information errors message that consists of the IFMP-C header. The Node Information errors message should be identical to the IFMP-C header of the Node Information request message except with the NACK flag set in Flags field 1020 and with Code field 1018 set to indicate the cause of the failure. Exemplary causes of failure include that the IFMP-C agent was unable to allocate a message buffer to complete the response, a client-specific error prevented the request from completing, or other causes.

Another IFMP-C Network Management message type, IFMP-C Interface Statistics messages are used to get information about each of the interfaces on the IFMP-C agent. An IFMP-C Interface Statistics request message 1460 allows the IFMP-C controller to request statistics for more than one interface in a single request message. As seen in Fig. 21b, an IFMP-C Interface Statistics request message 1460 has the generic format as shown in Fig. 15b with Message Body field 1030 that includes: a 16-bit Reserved field 1462, a 16-bit Number of Interfaces field 1464, followed by multiple Interface Identifier fields 1466, 1468 and others. Reserved field 1462 is reserved for future use, and may be set to 0 by the sender and ignored by the receiver. Number of Interfaces field 1464 indicates the number of Interface Identifier fields in the request message. Interface Statistics request

message 1460 includes multiple Interface Identifier fields (e.g. 1466 and 1468) to list identifiers for each interface the IFMP-C controller is interested in. In the specific embodiment, the reply must fit in a single message, so that the IFMP-C agent returns responses for as many interfaces that fit in a single response message.

If the IFMP-C Interface Statistics request message has the PLEASE_ACK flag set in its Flags field 1020 and the operation is successful, the IFMP-C agent returns an IFMP-C Interface Statistics response message 1470 (seen in Fig. 21c) with its Flags field 1020 set to an ACK flag and Code field 1018 set to the predetermined value (0 in the specific embodiment) indicating no error. As shown in Fig. 21c, IFMP-C Interface Statistics response message 1470 has the generic format as previously described in relation to Fig. 15b, with Message Body field 1030 containing: Interface Statistics fields (e.g., 1472, 1474) which provide general statistics (statistics common across different interface types) about the interface, and specific statistics (statistics that only apply to specific interface types). A typical Interface Statistics field structure 1480, as shown in Fig. 21d, includes: a 8-bit Reserved field 1482, a 8-bit Interface Type field 1484, a 16-bit Record Length field 1486, a 32-bit Interface Identifier field 1488, a 16-bit General Statistics Length field 1490, a Specific Statistics Length field 1492, a General Statistics field 1494, and a Specific Statistics field 1496. Reserved field 1482 is reserved for future use and may be set to 0 by the sender and ignored by the receiver. Interface Type field 1484 describes the type of interface (e.g. ATM, Ethernet, or other LAN interface) being queried. Interface Identifier field 1488 uniquely indicates which interface the statistics describe. General Statistics Length field 1490 specifies the length of the General Statistics field 1494 in octets, and Specific Statistics Length field 1492 specifies the length of the Specific Statistics field 1496 in octets. General Statistics field 1494 contains general statistics associated with all interfaces and has the structure shown in Fig. 21e. Specific Statistics field 1496 contains the interface type specific statistics, with exemplary structures (for ATM and Ethernet interfaces) shown in Figs. 21f and 21g. Other structures may be used for other LAN interfaces.

As seen in Fig. 21e, General Statistics field 1494 includes: a 64-bit Received Octets field 1500 which indicates the number of octets received on the

specified interface; a 64-bit Received Multicast Packets field 1502 which indicates the number of packets addressed as multicast packets that were received on the specified interface; a 64-bit Received Broadcast Packets field 1504 which indicates the number of packets addressed as broadcast packets that were received on the specified interface; a 64-bit Received Unicast Packets field 1506 which indicates the number of packets addressed as unicast packets that were received on the specified interface; a 64-bit Received Discards field 1508 which indicates the number of packets that were discarded on input for the specified interface; a 64-bit Received Errors field 1510 which indicates the number of receive errors on the specified interface; a 64-bit Received Unknown field 1512 which indicates the number of received packets having an unrecognized protocol on the specified interface; a 64-bit Transmit Octets field 1514 which indicates the number of octets transmitted on the specified interface; a 64-bit Transmit Multicast Packets field 1516 which indicates the number of packets addressed as multicast packets that were received on the specified interface; a 64-bit Transmit Broadcast Packets field 1518 which indicates the number of packets addressed as broadcast packets that were transmitted on the specified interface; a 64-bit Transmit Unicast Packets field 1520 which indicates the number of packets addressed as neither multicast nor broadcast packets that were transmitted on the specified interface; a 64-bit Transmit Discards field 1522 which indicates the number of packets that were discarded while transmitting on the specified interface; and a 64-bit Transmit Errors field 1524 which indicates the number of transmit errors that have occurred on the specified interface.

Specific Statistics field 1496 for specific statistics for ATM and Ethernet interfaces are respectively shown in Figs. 21f and 21g. As seen in Fig. 21f, Specific Statistics field structure 530 for an ATM interface includes: a 64-bit Received Cells field 1532 which indicates the number of ATM cells received on the specified interface; a 64-bit Transmit Cells field 1534 which indicates the number of ATM cells transmitted on the specified interface; a 64-bit AAL5 CRC Errors field 1535 which indicates the number of packets with incorrect AAL5 Cyclic Redundancy Checksum (CRC) received on the specified interface; and a 64-bit Physical Errors field 1536 which indicates the number of physical errors that have occurred on the specified ATM interface. Another example of Specific Statistics

field 1496 for an Ethernet interface includes: a 64-bit Received CRC Errors field 1542 which indicates the number of packets that have been received with an improper CRC on the specified interface; and a 64-bit Transmit Collisions field 1544 which indicates the number of collisions generated while attempting to send packets on the specified interface.

If the IFMP-C Interface Statistics request failed, then the IFMP-C agent returns an IFMP-C Interface Statistics errors message that consists of the IFMP-C header. The Interface Statistics errors message should be identical to the IFMP-C header of the Interface Statistics request message except with the NACK flag set in Flags field 1020 and with Code field 1018 set to indicate the cause of the failure. Exemplary causes of failure include that the interface identifier is invalid, the number of interfaces listed is not consistent with the number of interfaces in the message, the IFMP-C agent was unable to allocate a message buffer to complete the response, a client-specific error prevented the request from completing, or other causes.

IV. Conclusion

The inventions claimed herein provide an improved method and apparatus for transmitting packets over a network. It is to be understood that the above description is intended to be illustrative and not restrictive. Many embodiments will be apparent to those of skill in the art upon reviewing the above description. By way of example the inventions herein have been illustrated primarily with regard to transmission of IP packets capable of carrying voice, video, image, facsimile, and data signals, but they are not so limited. By way of further example, the invention has been illustrated in conjunction with specific components and operating speeds, but the invention is not so limited. Still further, it is understood that although specific examples of message types, errors, etc. are described for a specific embodiment, these are merely examples and other embodiments can use different, additional, fewer, or a combination of the features described. The scope of the inventions should, therefore, be determined not with reference to the above description, but should instead be determined with reference

to the appended claims, along with the full scope of equivalents to which such claims are entitled, by one of ordinary skill in the art.

WHAT IS CLAIMED IS:

1. A method for transmitting packets between an upstream node and a downstream node in a network, said downstream node being downstream from said upstream node, said method comprising the steps of:

establishing default virtual channels between said upstream node and said downstream node;

receiving a packet at said downstream node;

performing a flow classification at said downstream node on said packet to determine whether said packet belongs to a specified flow that should be redirected in the upstream node;

selecting a free label at said downstream node;

informing said upstream node that future packets belonging to said specified flow should be sent with said selected free label attached.

2. The method of claim 1 wherein said upstream and downstream nodes use ATM.

3. The method of claim 2 wherein said free label comprises a VPI/VCI.

4. The method of claim 1 wherein said network comprises a local area computer network.

5. The method of claim 4 wherein informing step is performed by IFMP software that enables communication between said upstream and downstream nodes.

6. The method of claim 1 wherein said flow classification performing step includes looking at a flow identifier of the packet to determine whether said packet belongs to a specified flow that should be redirected in the upstream node, said flow identifier comprising specified header fields from the packet; and

sending said future packets belonging to the specified flow with said specified header fields removed to provide security.

7. The method of claim 6 wherein said future packets may be reconstructed using the flow identifier at a destination permitted to receive said specified flow.

8. A method for switching a flow at a first node, said first node having a downstream link to a second node and an upstream link to a third node, said method comprising the steps of:

performing a flow classification at said first node on a first packet to determine whether said first packet belongs to a specified flow that should be redirected in the third node;

selecting a first free label at said first node;

informing said third node that future packets belonging to said specified flow should be sent with said selected first free label attached;

performing a flow classification at said second node on a second packet to determine whether said second packet belongs to said specified flow that should be redirected in the third node;

selecting a second free label at said second node;

informing said first node that future packets belonging to said specified flow should be sent with said selected second free label attached; and

wherein said specified flow from said upstream link may be switched in layer 2 by said first node to said downstream link.

9. The method of claim 8 wherein said second packet is said first packet.

10. The method of claim 9 wherein said first, second and third nodes use ATM.

11. The method of claim 10 wherein said first and second free labels comprise VPI/VCIs.

12. The method of claim 9 wherein said informing steps are performed by IFMP software that enables communication between said nodes.

13. A computer program product that enables dynamic shifting between routing and switching in a network having an upstream node and a downstream node downstream from said upstream node, said computer program product comprising:

computer readable code that performs a flow classification on a packet at said downstream node to determine whether said packet belongs to a specified flow that should be redirected in said upstream node;

computer readable code that selects a first free label at said downstream node;

computer readable code that informs said upstream node that future packets belonging to said specified flow should be sent with said selected first free label attached; and

a tangible medium that stores the computer readable codes.

14. The computer program product of claim 13, wherein said tangible media comprises a hard disk on a computer.

15. The computer program product of claim 13, wherein said tangible media is selected from a group consisting of CD-ROM, tape, floppy disk, and the like.

16. A basic switching unit in a system for transmitting packets in a network, said basic switching unit comprising:

a switching hardware;

a controller coupled to said switching hardware, wherein said controller includes a processor and memory, said controller controlling said switch; and

software, said software fixed on tangible media, wherein said software enables the basic switching unit to dynamically shift between packet routing and switching to optimize packet traffic throughput.

17. The basic switching unit of claim 16 wherein said software utilizes flow classification.

18. The basic switching unit of claim 17 wherein said switching hardware utilizes asynchronous transfer mode (ATM) switching technology.

19. The basic switching unit of claim 18 wherein said flow classification uses VPI/VCI as labels.

20. The basic switching unit of claim 18 wherein said software includes a first software subset installed on said controller to communicate with and control said switching hardware.

21. The basic switching unit of claim 20 wherein said software further includes a second software subset enabling communication between two of said basic units and defining the format for flow redirect messages and acknowledgements.

22. The basic switching unit of claim 21 wherein said basic unit locally makes flow classification decisions and response to redirect message decisions.

23. The basic switching unit of claim 16 wherein said network comprises an area network including computers.

24. The basic switching unit of claim 20 wherein said first software subset comprises IFMP.

25. The basic switching unit of claim 22 wherein said second software subset comprises GSMP.

26. The basic switching unit of claim 18 wherein said software provides quality of service capability.

27. The basic switching unit of claim 17 wherein said switch utilizes fast packet technology.

28. The basic switching unit of claim 17 wherein said switch utilizes frame relay technology.

29. The basic switching unit of claim 17 wherein said switch utilizes Gigabit Ethernet technology.

30. A switch gateway unit in a system for transmitting packets in a network, said system including a basic switching unit coupled to said switch gateway unit via a communication link, said switch gateway unit comprising:

a gateway controller, said gateway controller including a processor, memory, and a plurality of NICs;

software, said software fixed on tangible media, wherein said software enables the switch gateway unit to redirect a flow of packets to said basic switching unit to enable dynamic shifting between packet routing and switching to optimize packet traffic throughput.

31. The switch gateway unit of claim 30 wherein said software utilizes flow classification.

32. The switch gateway unit of claim 30 wherein said basic switching unit utilizes asynchronous transfer mode (ATM) switching technology.

33. The switch gateway unit of claim 32 wherein said switch gateway unit and said basic switching unit use VPI/VCI as labels.

34. The switch gateway unit of claim 33 wherein said software includes a first software subset installed on said gateway controller, said first software subset enabling communication between said switch gateway unit and said basic switching unit in said system and defining the format for flow redirect messages and acknowledgements.

35. The switch gateway unit of claim 34 wherein said switch gateway unit locally makes flow classification decisions and responds to redirect message decisions.

36. The switch gateway unit of claim 35 wherein said first software subset comprises IFMP.

37. The switch gateway unit of claim 30 wherein said basic switching unit utilizes Gigabit Ethernet technology.

38. A switching agent in a system for transmitting packets in a network, said system including a basic switching unit coupled to said switching agent via a communication link, said basic switching unit including a controller and a switching engine, said switching agent comprising:

a processor, a memory, and a plurality of NICs, a specific one of said plurality of NICs providing said communication link and at least one of said plurality of NICs connectable to at least one node in said network; and

computer-readable program code, said computer-readable program code fixed on a tangible computer-readable media comprising said memory, wherein said computer-readable program code enables the controller of said basic switching unit to classify a flow and to redirect said flow of packets from a first node to a second node in said network, and wherein said computer-readable program code enables said controller of said basic switching unit to instruct said switching agent to

perform packet forwarding of said flow from said first node to said second node via said switching engine, thereby offloading packet forwarding from said controller of said basic switching unit.

39. The switching agent of claim 38 wherein said first node is connected via a first one of said plurality of NICs to said switching agent and said second node is selected from the group consisting of another of said switching agent, another of said basic switching unit, a switch gateway unit, or host; and wherein said second node is coupled to said switching engine of said basic switching unit.

40. The switching agent of claim 38 wherein said first node is selected from the group consisting of another of said switching agent, another of said basic switching unit, a switch gateway unit, or host; and wherein said second node is connected via a first one of said plurality of NICs to said switching agent; and wherein said computer-readable program code enables said controller of said basic switching unit to instruct said switching agent on how to handle said packets in said flow received from said switching engine.

41. The switching agent of claim 38 wherein said switching engine utilizes asynchronous transfer mode (ATM), frame relay, fast packet switching, 10 Mbps Ethernet, 100 Mbps Ethernet, or Gigabit Ethernet technology.

42. The switching agent of claim 38 wherein at least one of said plurality of NICs is an Ethernet NIC.

43. The switching agent of claim 38 wherein at least one of said plurality of NICs is an ATM NIC.

44. The switching agent of claim 38 wherein said computer-readable program code includes a first subset installed on said controller and a second subset installed on said memory of said switching agent, said first subset and said second

enabling communication between said switching agent and said basic switching unit in said system.

45. The switching agent of claim 44 wherein said switching agent serves as a slave to said basic switching unit which locally makes flow classification decisions and responds to redirect message decisions.

46. The switching agent of claim 44 wherein said computer-readable program code comprises IFMP-C protocol software.

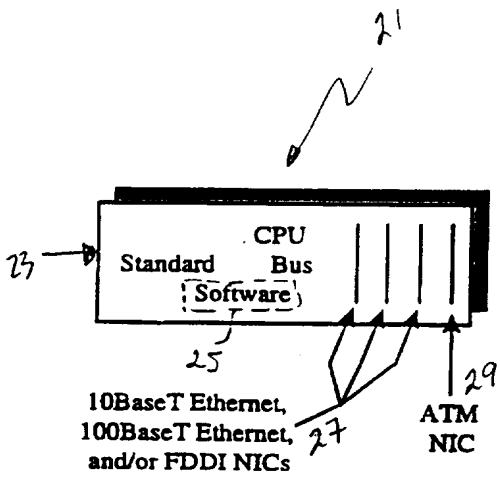


FIG. 1b

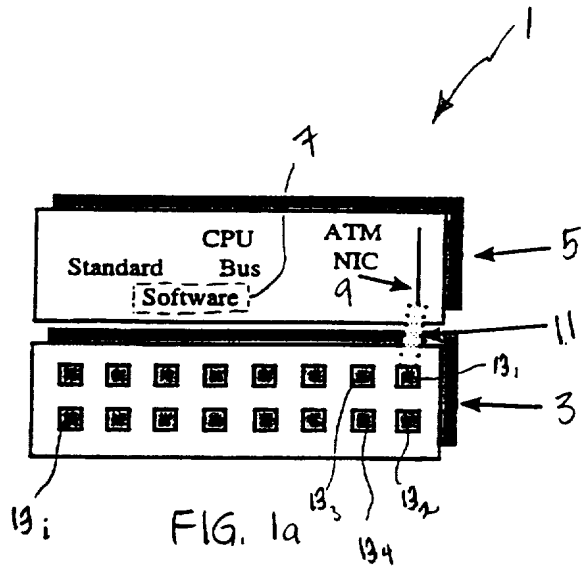


FIG. 1a

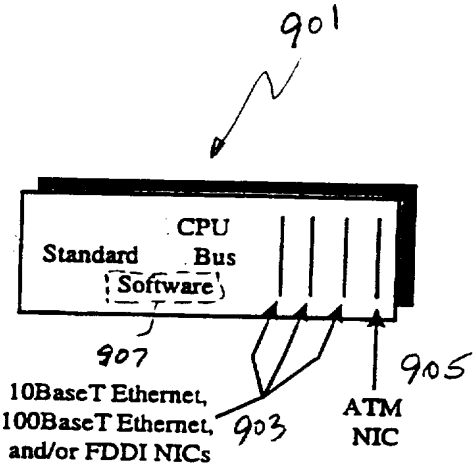


Fig. 1c

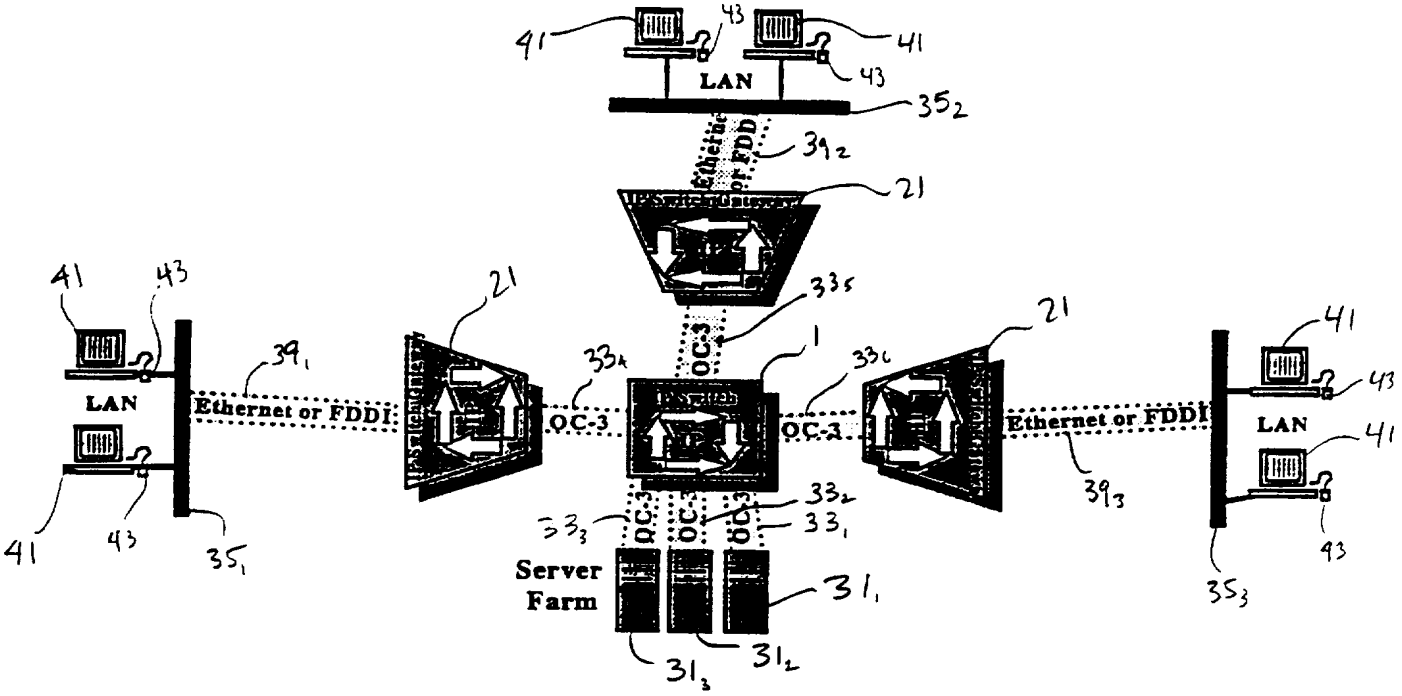

LAN
Ethernet or FDDI
OC-3
Server Farm
LAN
Ethernet or FDDI
OC-3
OC-3
OC-3
LAN
Ethernet or FDDI

FIG. 2a

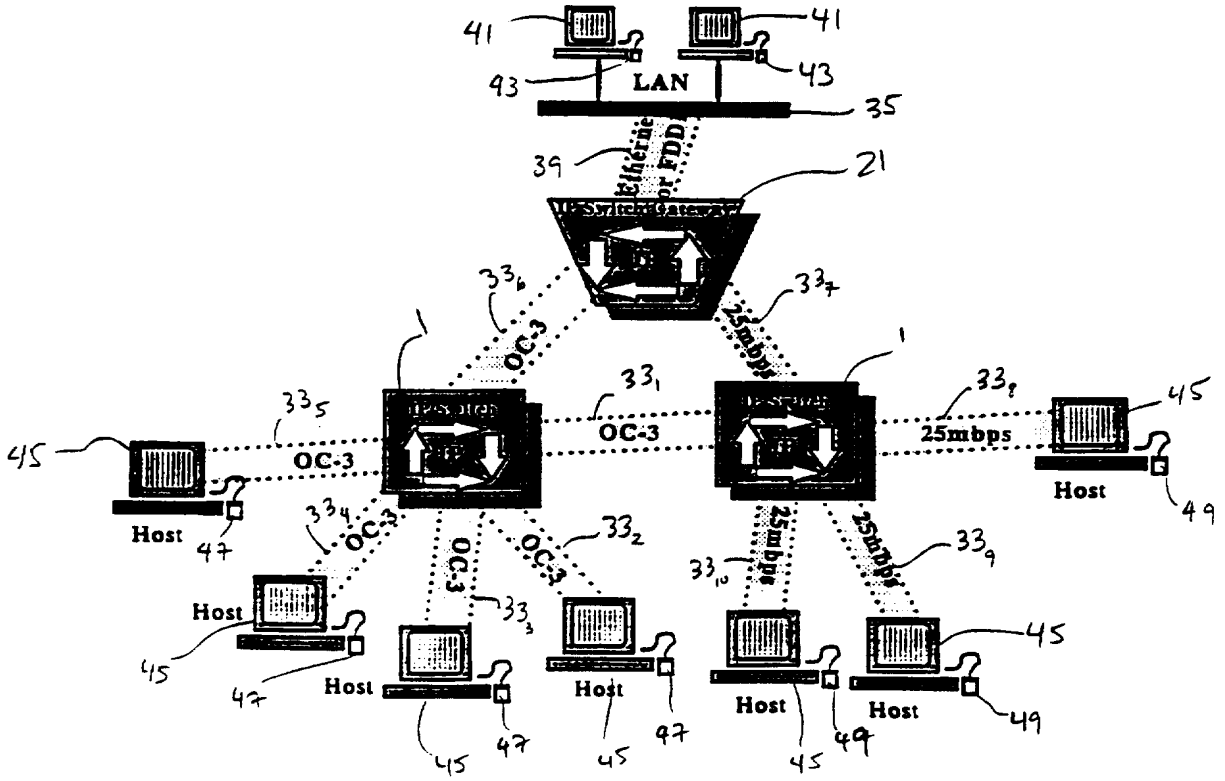

LAN
Ethernet or FDDI
OC-3
OC-3
25mbps
OC-3
25mbps
Host
Host
Host
Host
Host
Host
Host
Host
Host

FIG. 2b

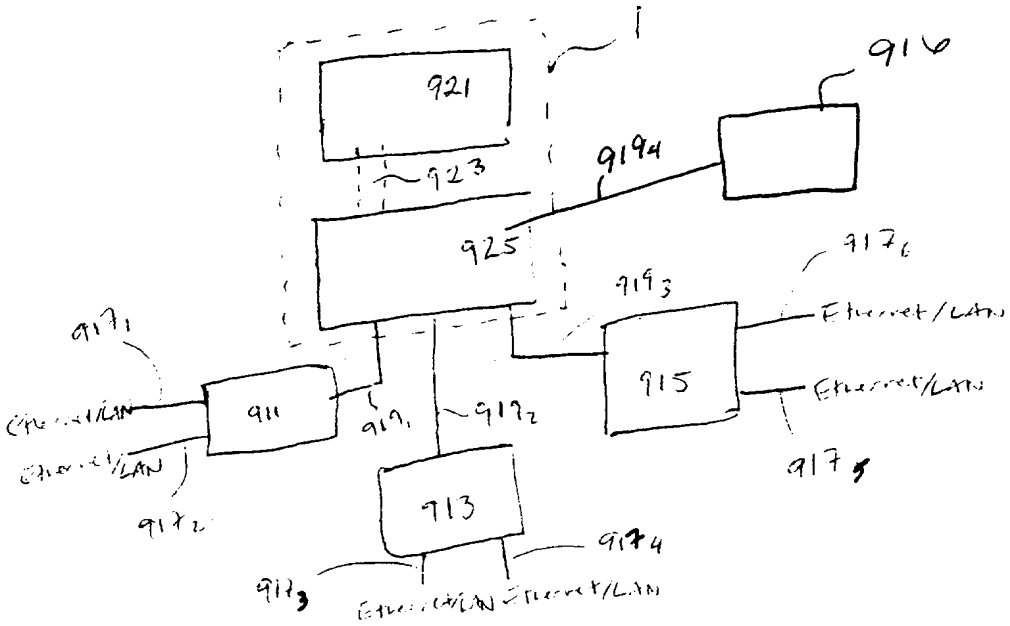

921
923
916
9194
925
9196
9193
Ethernet/LAN
9171
915
911
9191
9192
Ethernet/LAN
Ethernet/LAN
9195
9172
913
9174
9173
Ethernet/LAN
Ethernet/LAN

Fig. 2c

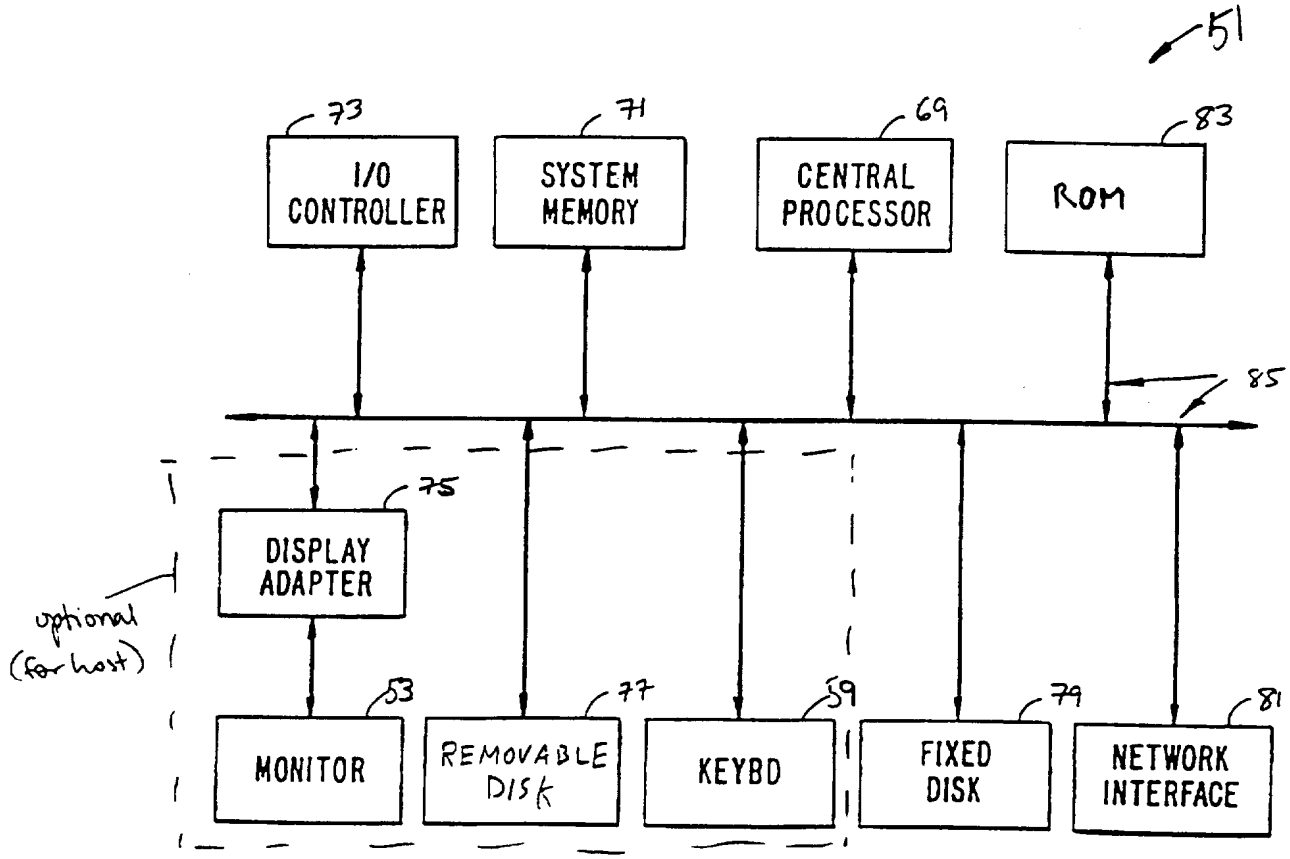

51
73
I/O CONTROLLER
71
SYSTEM MEMORY
69
CENTRAL PROCESSOR
83
ROM
85
75
DISPLAY ADAPTER
optional (for host)
53
MONITOR
77
REMOVABLE DISK
59
KEYBD
79
FIXED DISK
81
NETWORK INTERFACE

FIG. 3.

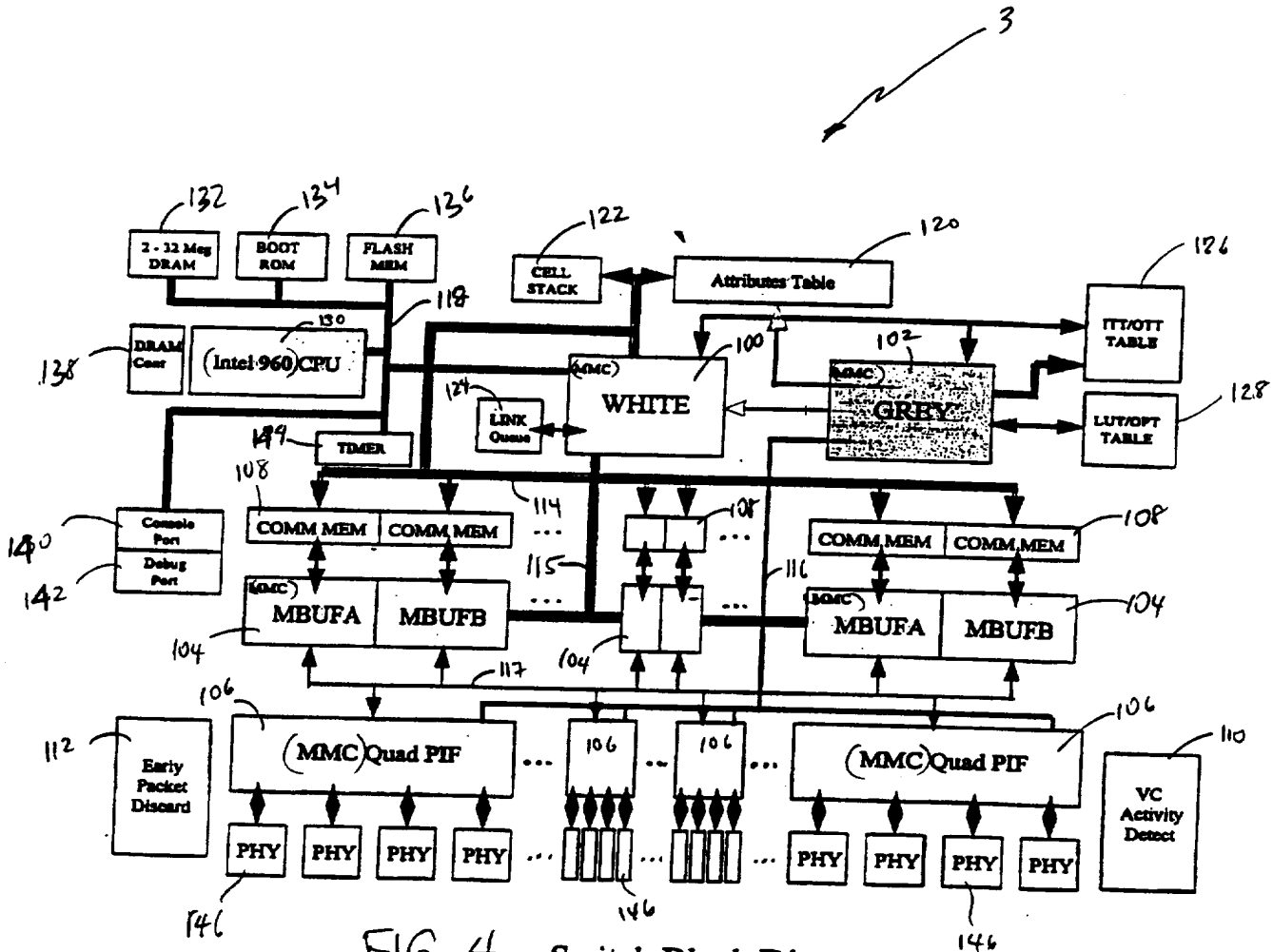


FIG. 4 Switch Block Diagram

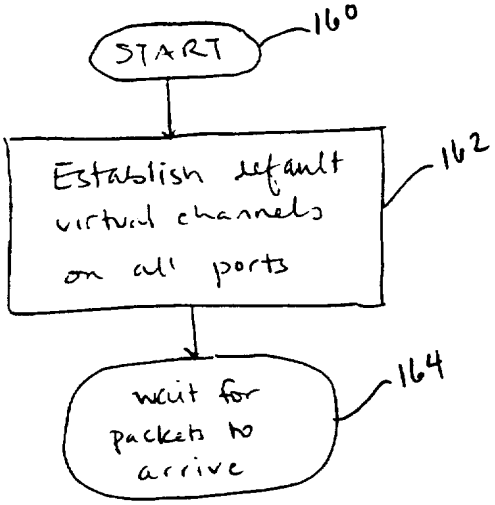

START
160
Establish default virtual channels on all ports
162
wait for packets to arrive
164

Fig. 5a

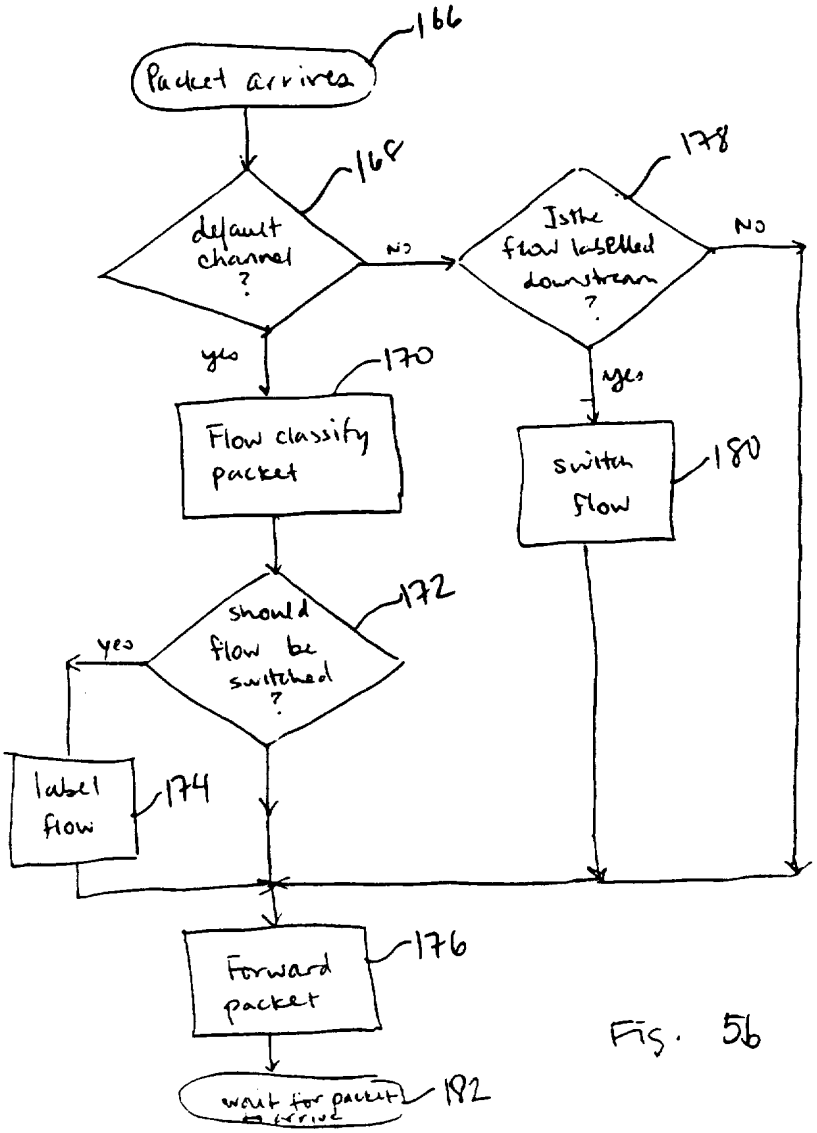

Packet arrives
166
default channel ?
168
NO
Is the flow labelled downstream ?
178
NO
yes
yes
Flow classify packet
170
switch flow
180
should flow be switched ?
172
yes
label flow
174
Forward packet
176
wait for packet to arrive
182

Fig. 5b

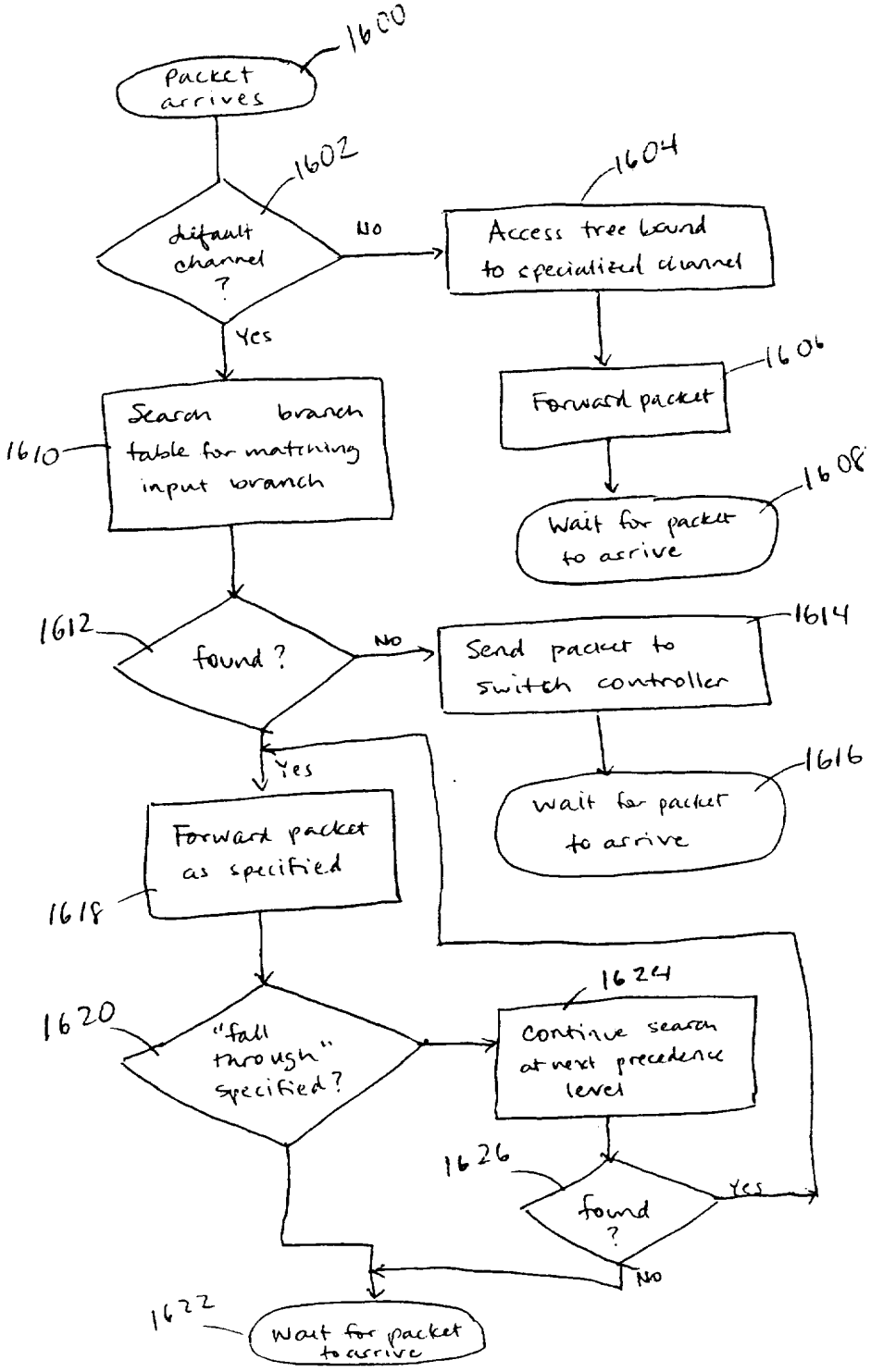

1600
Packet arrives
1602
default channel ?
No
Yes
1604
Access tree bound to specialized channel
1606
Forward packet
1608
Wait for packet to arrive
1610
Search branch table for matching input branch
1612
found ?
No
Yes
1614
Send packet to switch controller
1616
Wait for packet to arrive
Forward packet as specified
1618
1620
"fall through" specified?
1624
Continue search at next precedence level
1626
found ?
Yes
No
1622
Wait for packet to arrive

FIG. 5c

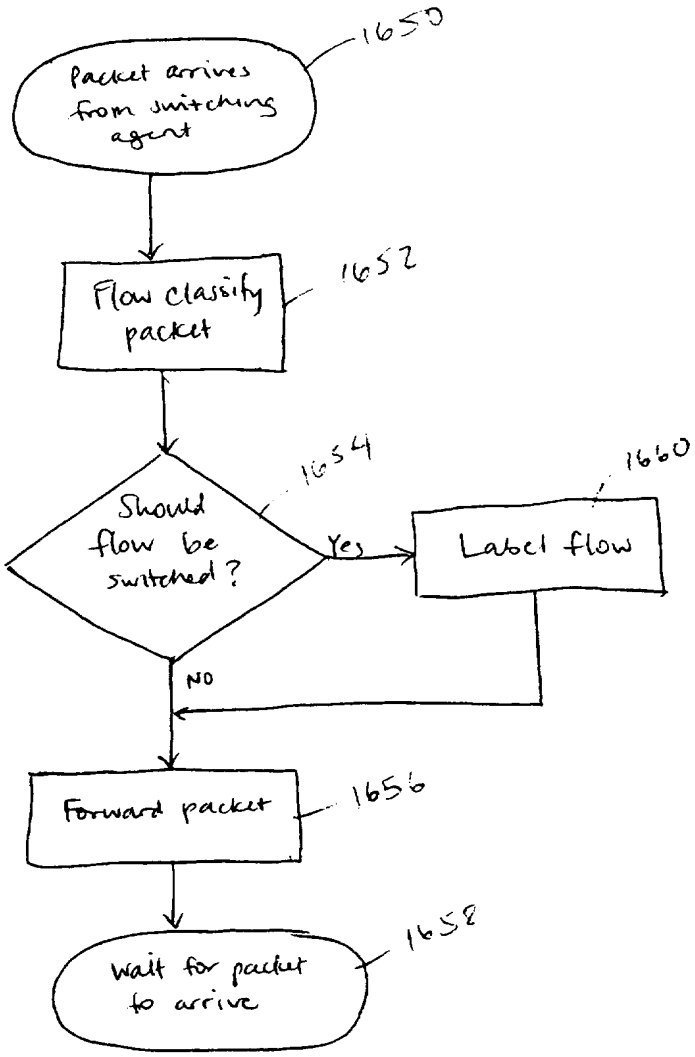

1650
Packet arrives from switching agent
1652
Flow classify packet
1654
Should flow be switched?
Yes
1660
Label flow
NO
Forward packet
1656
Wait for packet to arrive
1658

FIG. 5d

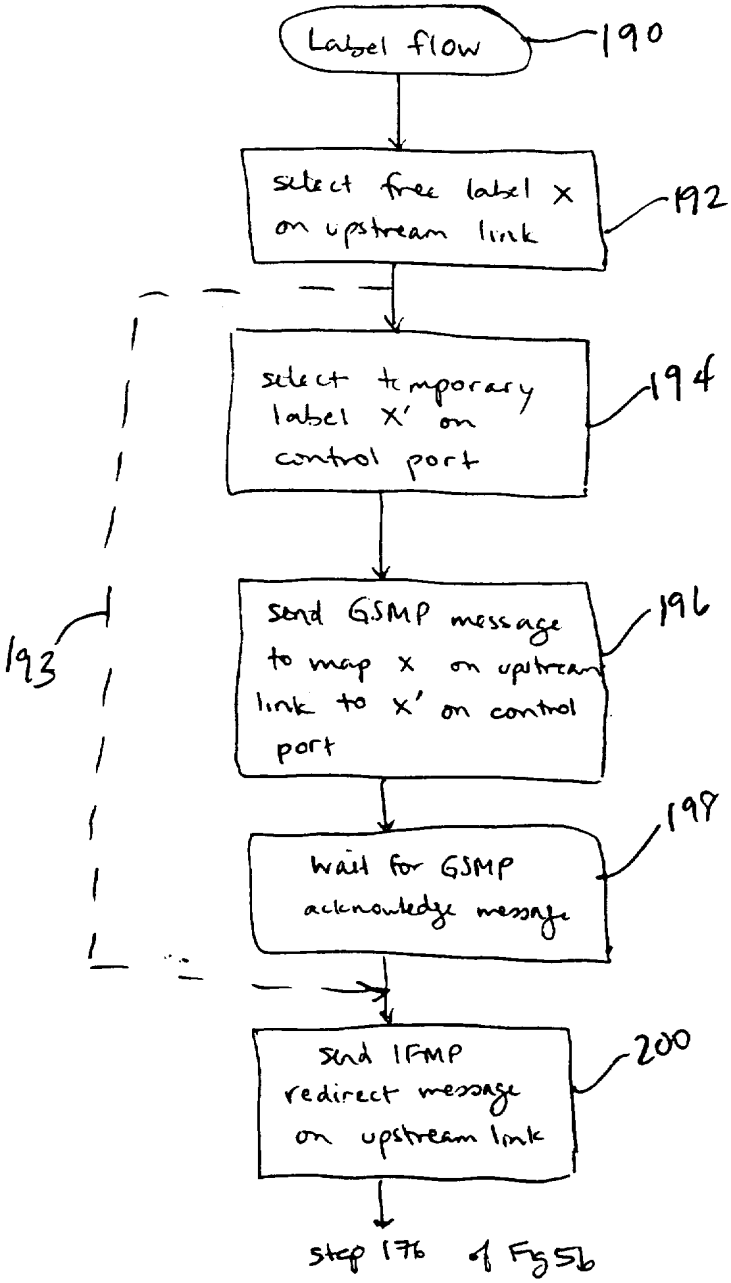

Label flow
190
select free label X on upstream link
192
select temporary label X' on control port
194
193
send GSMP message to map X on upstream link to X' on control port
196
wait for GSMP acknowledge message
198
send IFMP redirect message on upstream link
200
step 176 of Fig 5b

Fig. 6a

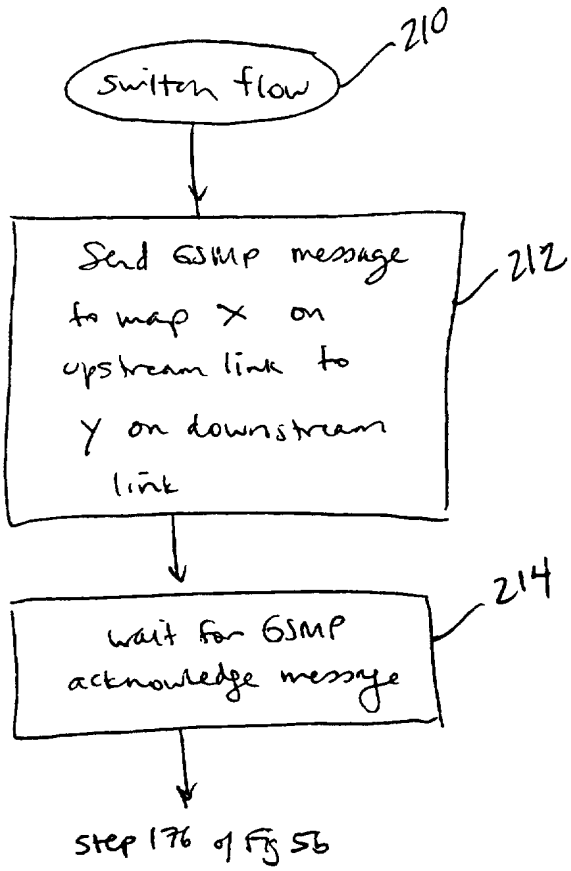

210
Switch flow
Send GSMP message to map X on upstream link to Y on downstream link
212
wait for GSMP acknowledge message
214
step 176 of Fig 5b

Fig. 6b

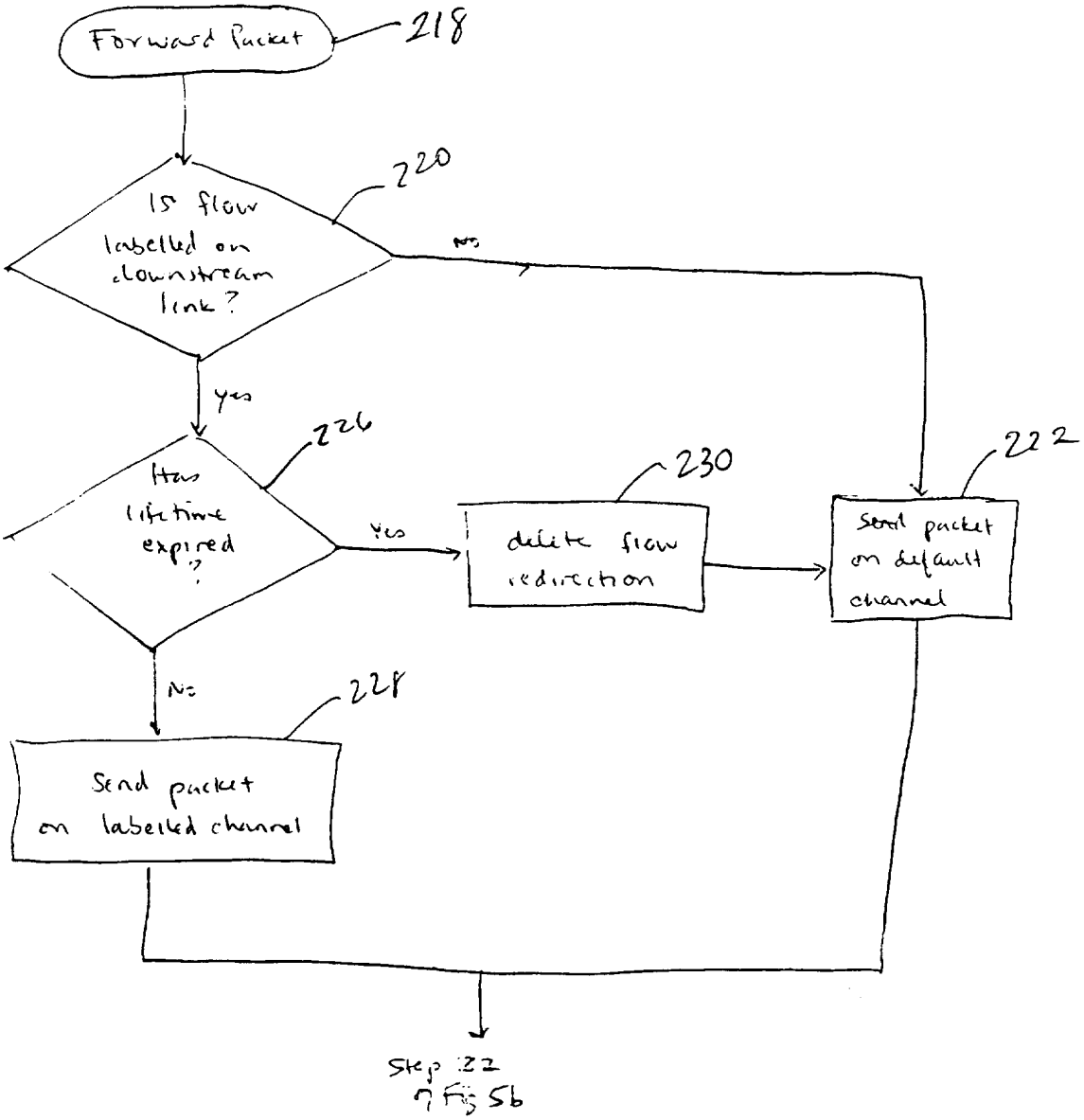

Forward Packet
218
Is flow labelled on downstream link?
220
No
Yes
Has lifetime expired?
226
Yes
No
delete flow redirection
230
Send packet on default channel
222
Send packet on labelled channel
228
Step 122 of Fig 5b

Fig. 6c

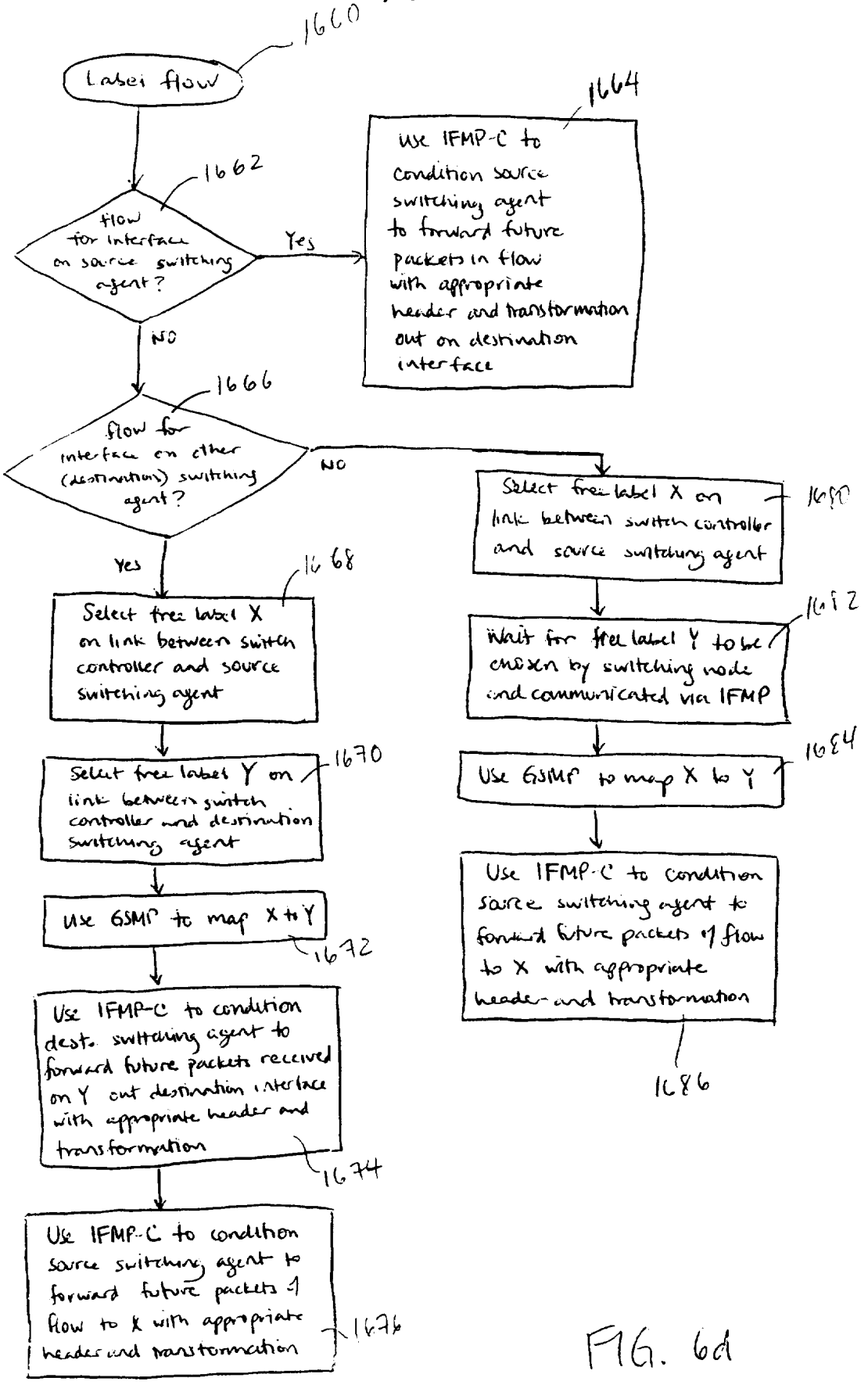

1660
Label flow
1662
flow for interface on source switching agent?
Yes
1664
Use IFMP-C to condition source switching agent to forward future packets in flow with appropriate header and transformation out on destination interface
NO
1666
flow for interface on other (destination) switching agent?
NO
Yes
1668
Select free label X on link between switch controller and source switching agent
1670
Select free label Y on link between switch controller and destination switching agent
Use GSMP to map X to Y
1672
Use IFMP-C to condition dest. switching agent to forward future packets received on Y out destination interface with appropriate header and transformation
1674
Use IFMP-C to condition source switching agent to forward future packets of flow to X with appropriate header and transformation
1676
1680
Select free label X on link between switch controller and source switching agent
1682
Wait for free label Y to be chosen by switching node and communicated via IFMP
1684
Use GSMP to map X to Y
Use IFMP-C to condition source switching agent to forward future packets of flow to X with appropriate header and transformation
1686

FIG. 6d

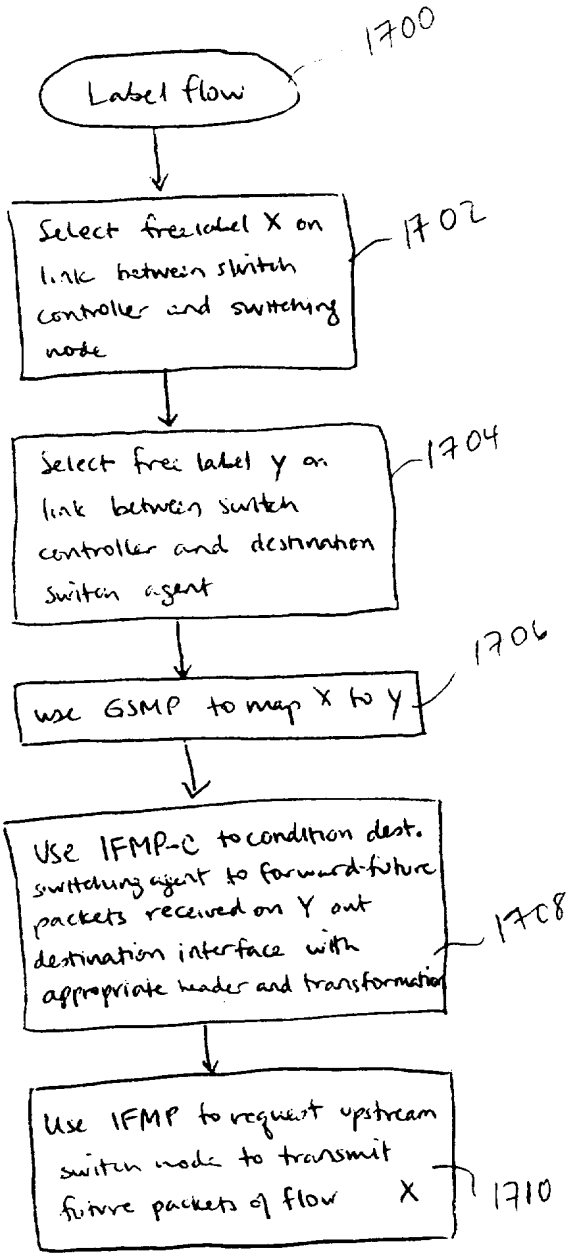

1700
Label flow
Select free label X on link between switch controller and switching node
1702
Select free label Y on link between switch controller and destination switch agent
1704
use GSMP to map X to Y
1706
Use IFMP-C to condition dest. switching agent to forward future packets received on Y out destination interface with appropriate header and transformation
1708
Use IFMP to request upstream switch node to transmit future packets of flow X
1710

Fig. 6e

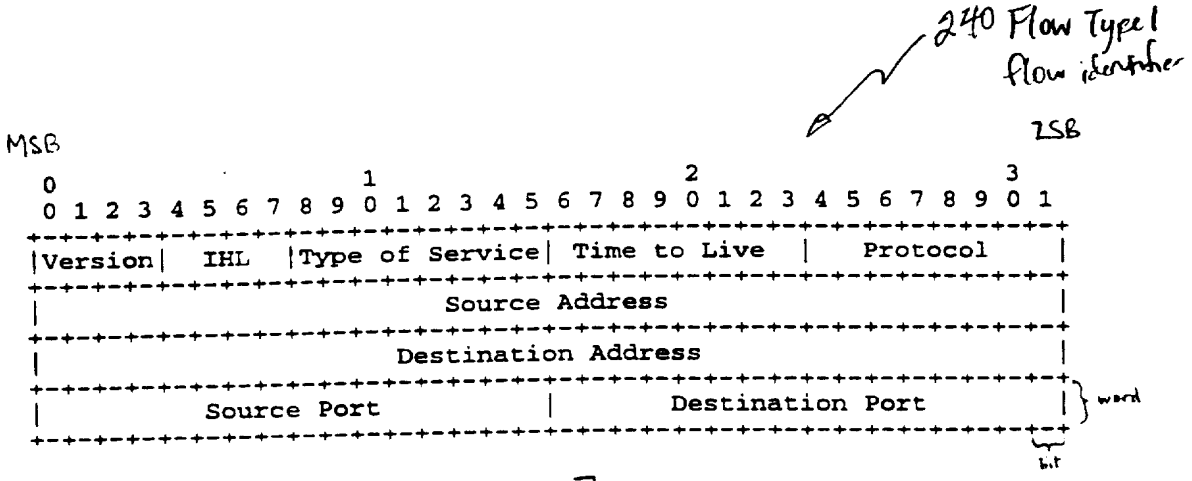

240 Flow Type/ flow identifier
MSB
LSB
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Version
IHL
Type of Service
Time to Live
Protocol
Source Address
Destination Address
Source Port
Destination Port
word
bit

FIG. 7a

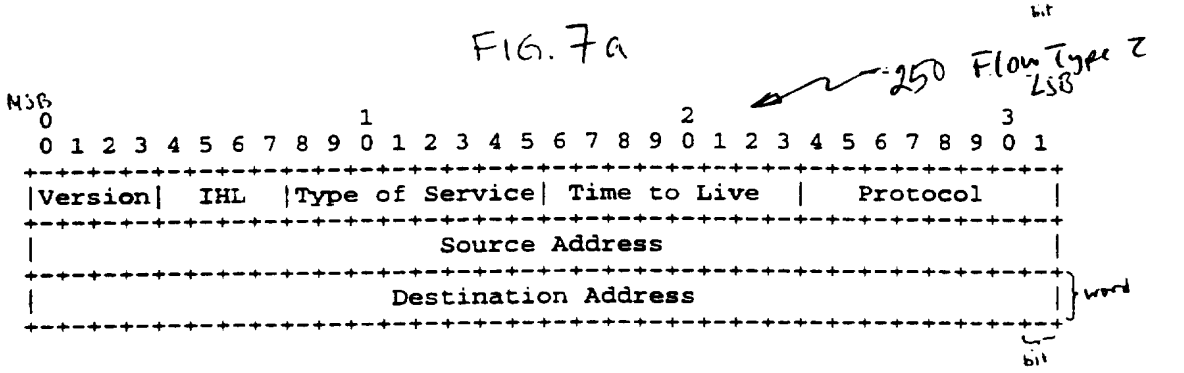

250 Flow Type 2 flow identifier
MSB
LSB
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Version
IHL
Type of Service
Time to Live
Protocol
Source Address
Destination Address
word
bit

FIG. 7b

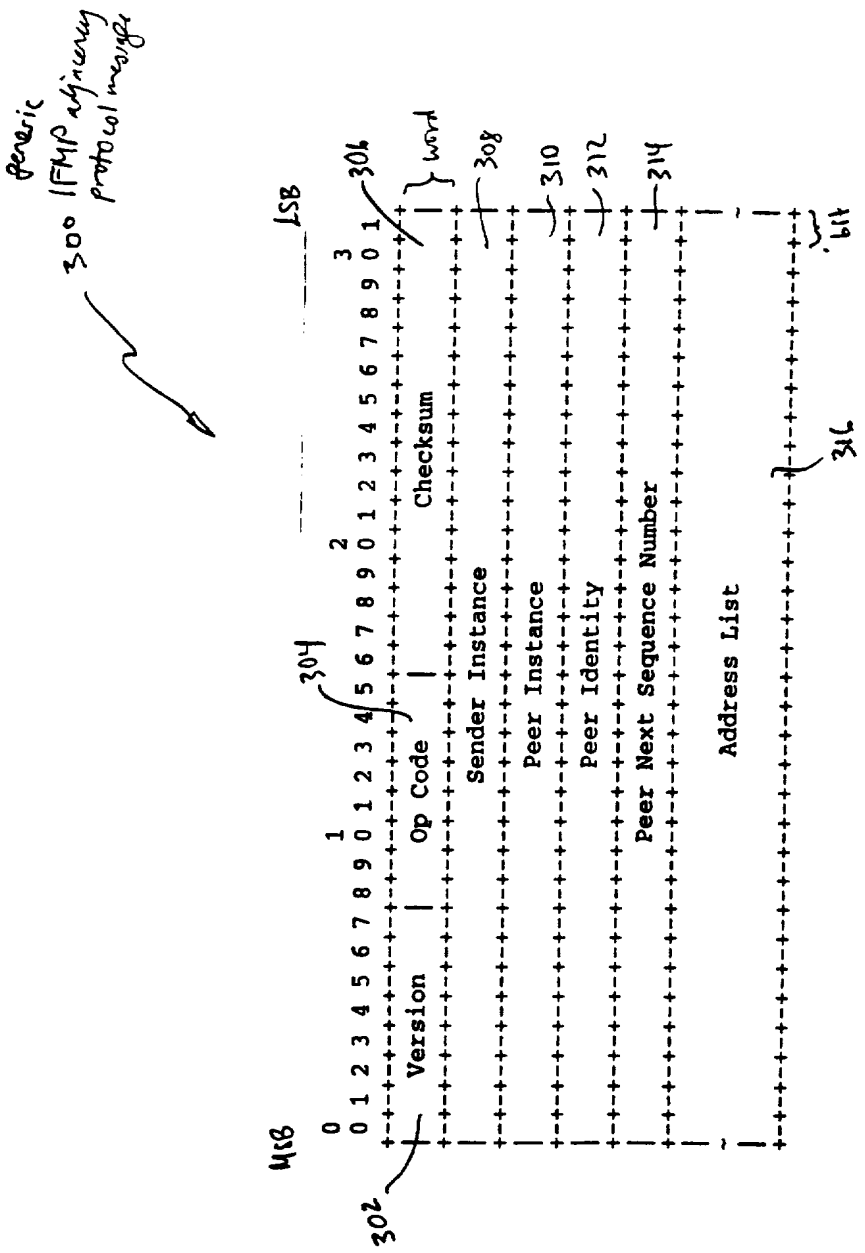
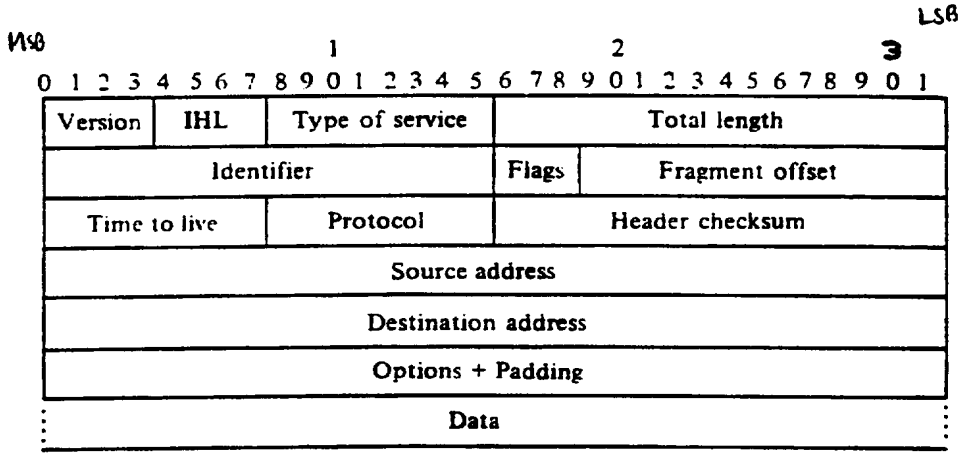

generic
300 IFMP shimmy protocol message
MSB
LSB
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Version
Op Code
Checksum
word
Sender Instance
Peer Instance
Peer Identity
Peer Next Sequence Number
Address List
302
304
306
308
310
312
314
316
318

FIG. 8a



generic IP packet (IPv4)

FIG. 8b

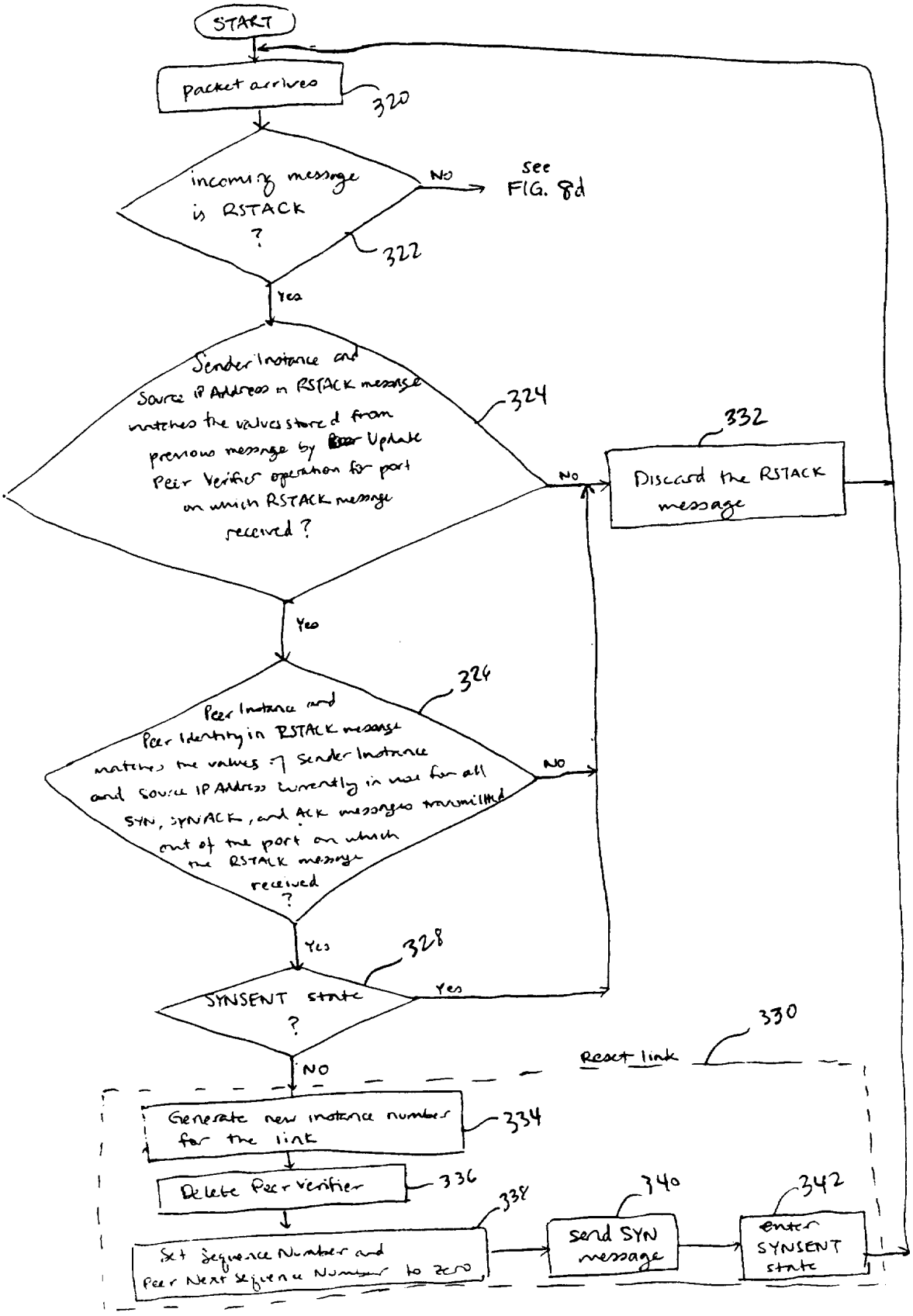
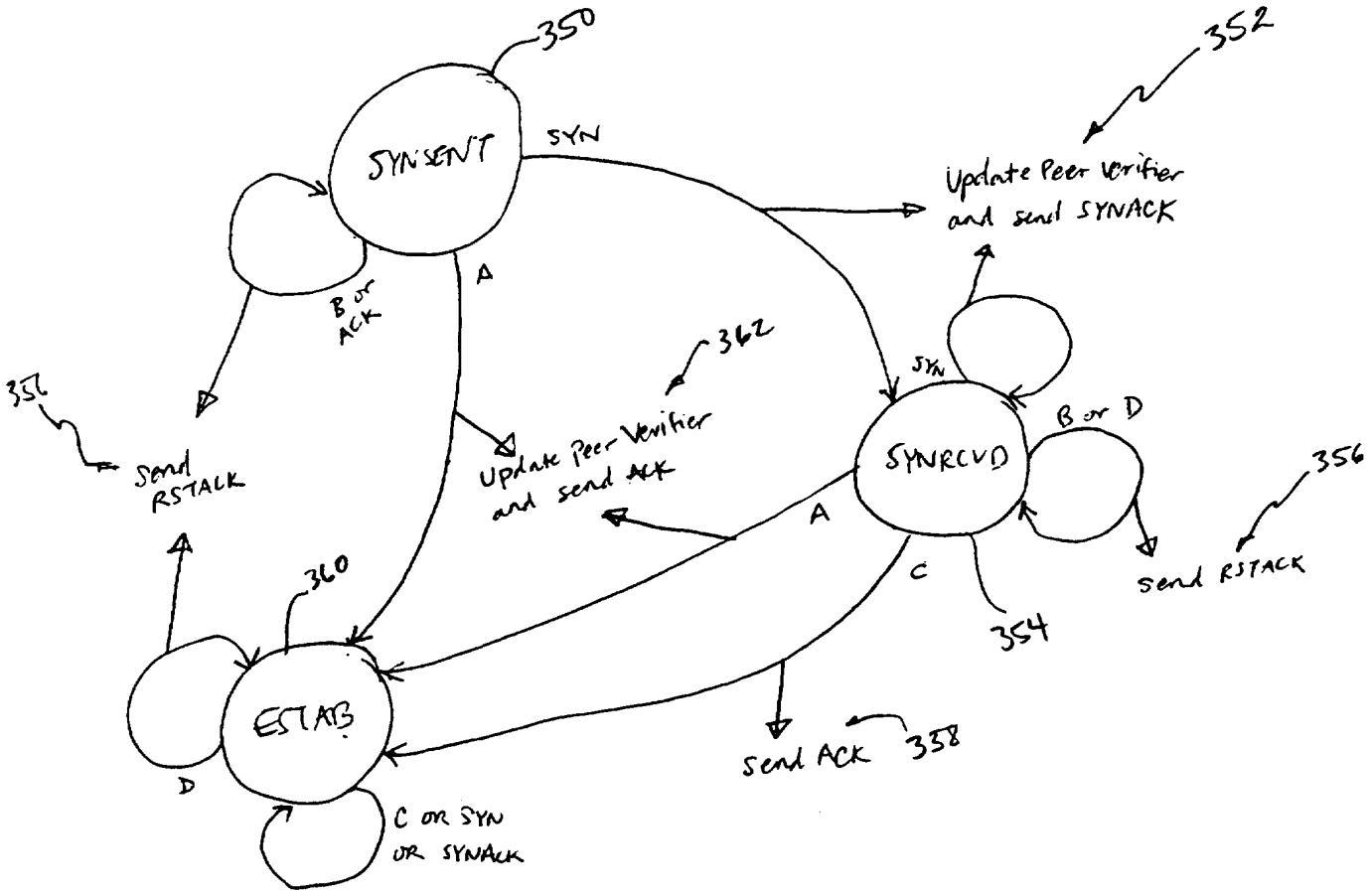

START
packet arrives
320
incoming message is RSTACK ?
No
see FIG. 8d
322
Yes
Sender Instance and Source IP Address in RSTACK message matches the values stored from previous message by Peer Update Peer Verifier operation for port on which RSTACK message received ?
324
No
332
Discard the RSTACK message
Yes
Peer Instance and Peer Identity in RSTACK message matches the values of Sender Instance and Source IP Address currently in use for all SYN, SYNACK, and ACK messages transmitted out of the port on which the RSTACK message received ?
324
No
Yes
328
SYNSENT state ?
Yes
NO
330
React link
Generate new instance number for the link
334
Delete Peer Verifier
336
Set Sequence Number and Peer Next Sequence Number to zero
338
Send SYN message
340
enter SYNSENT state
342

FIG. 8c

Fig. 8d



Legend

A: SYNACK AND %C

B: SYNACK AND NOT(%C)

C: ACK AND %B AND %C

D: ACK AND NOT(%B AND %C)

where %B: Sender Instance and Source IP Address in incoming message matches the values stored from previous message by Update Peer Verifier Operation for the port on which incoming message is received

where %C: Peer Instance and Peer Identity in incoming message matches the values of Sender Instance and Source IP Address currently in use for all SYN, SYNACK, and ACK messages transmitted out of the port on which incoming message is received

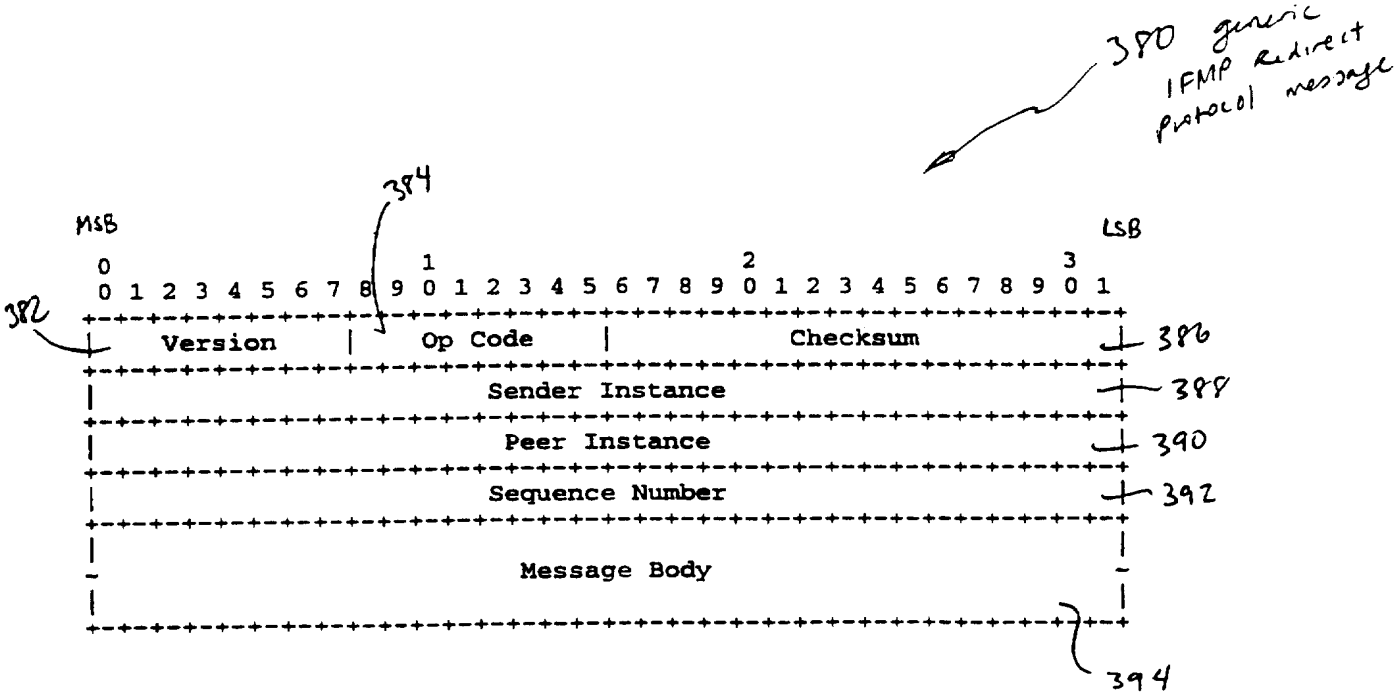

380 generic IFMP Redirect protocol message
384
MSB
LSB
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
382
Version
Op Code
Checksum
386
Sender Instance
388
Peer Instance
390
Sequence Number
392
Message Body
394

FIG. 9a

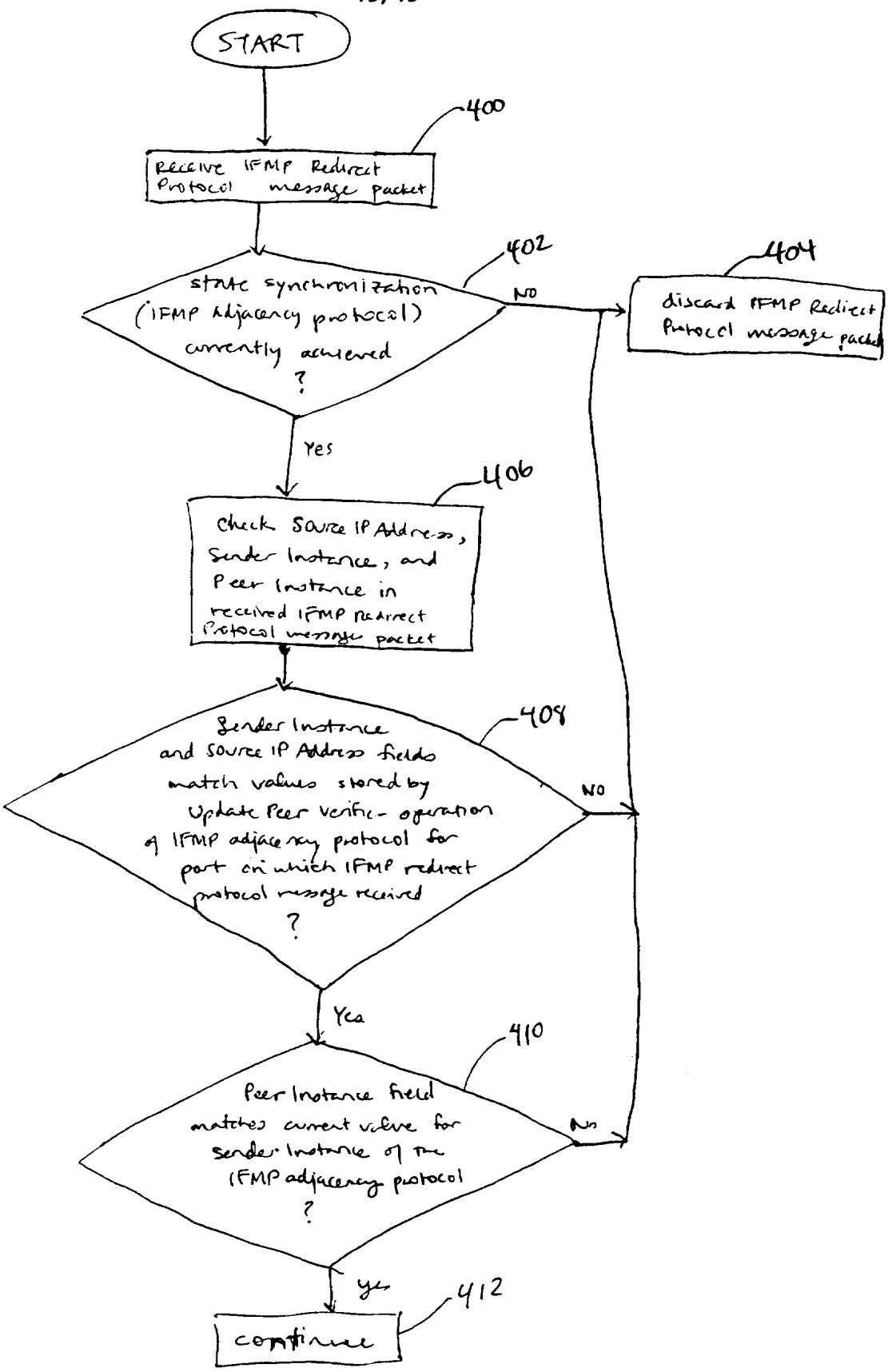

START
400
Receive IFMP Redirect Protocol message packet
402
state synchronization (IFMP Adjacency protocol) currently achieved ?
NO
404
discard IFMP Redirect Protocol message packet
Yes
406
Check Source IP Address, Sender Instance, and Peer Instance in received IFMP Redirect Protocol message packet
408
Sender Instance and Source IP Address fields match values stored by Update Peer verification operation of IFMP adjacency protocol for port on which IFMP redirect protocol message received ?
NO
Yes
410
Peer Instance field matches current value for sender Instance of the IFMP adjacency protocol ?
No
Yes
412
continue

FIG. 9b

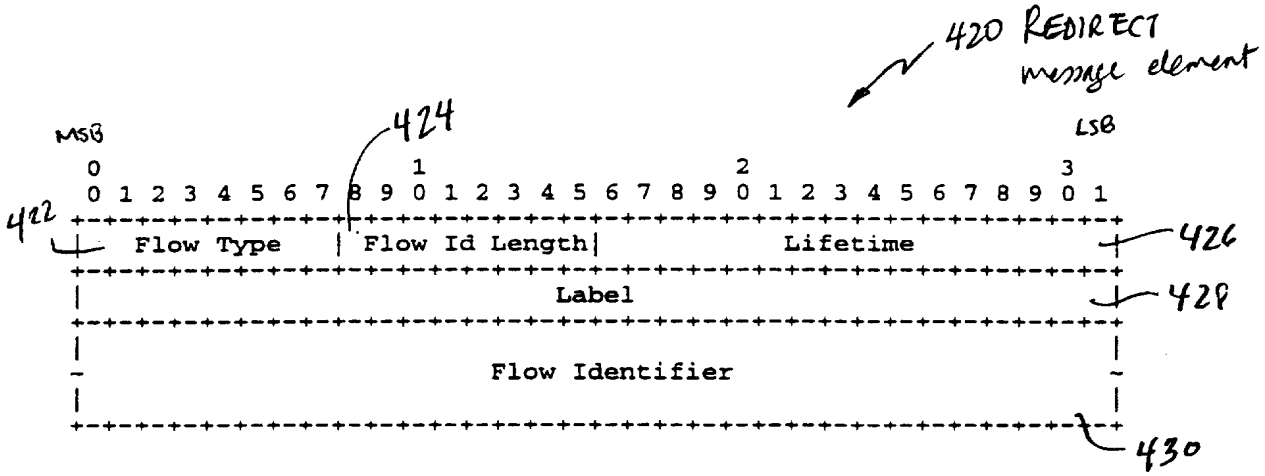


FIG. 9c

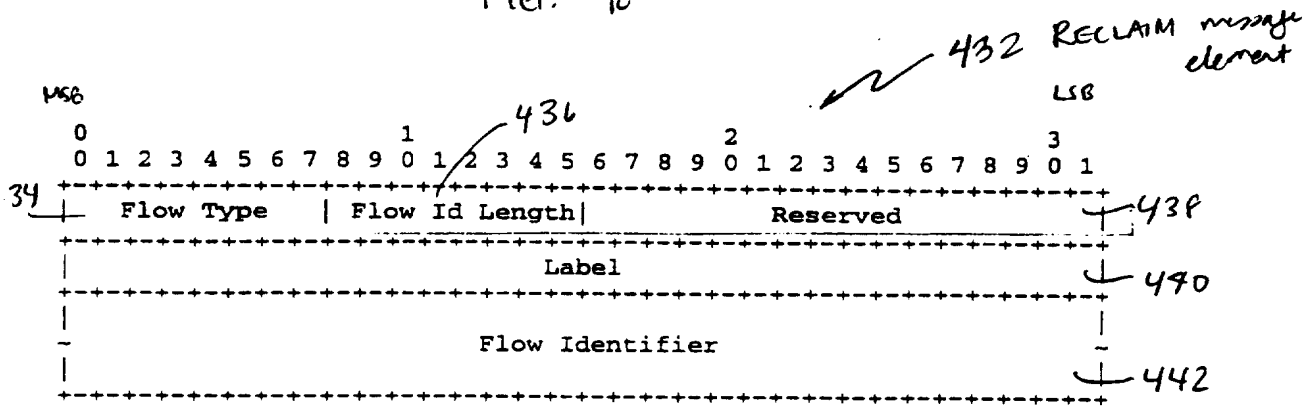


FIG. 9d

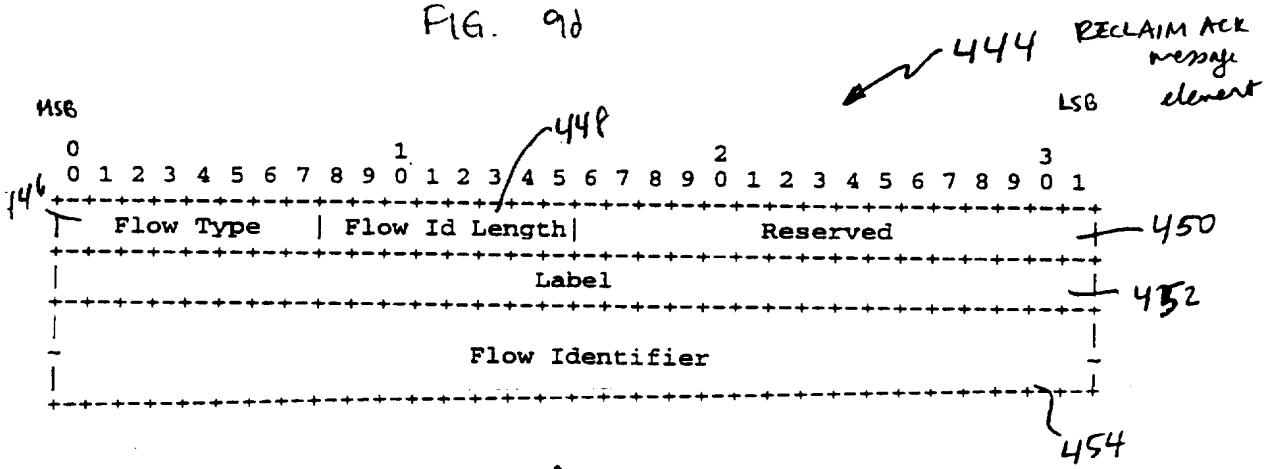


FIG. 9e

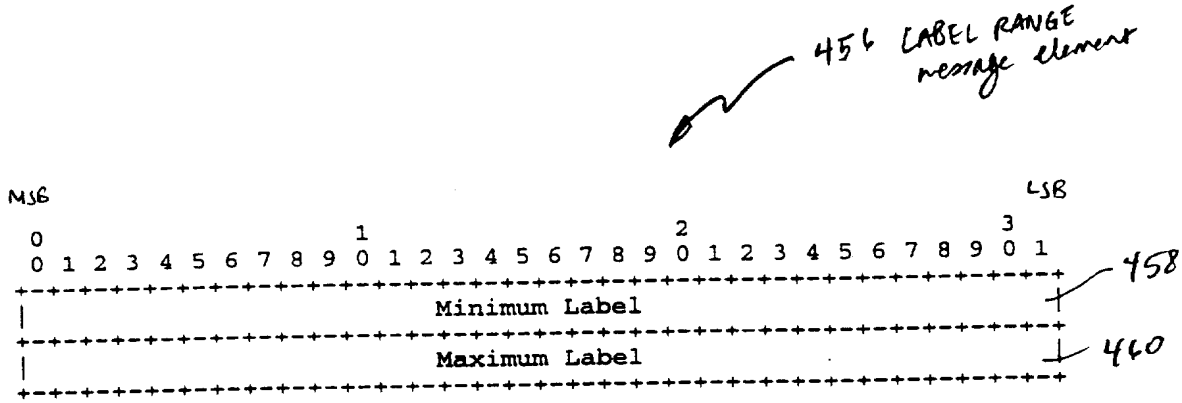

456 LABEL RANGE message element
MSB
LSB
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Minimum Label
Maximum Label
458
460

FIG. 9f

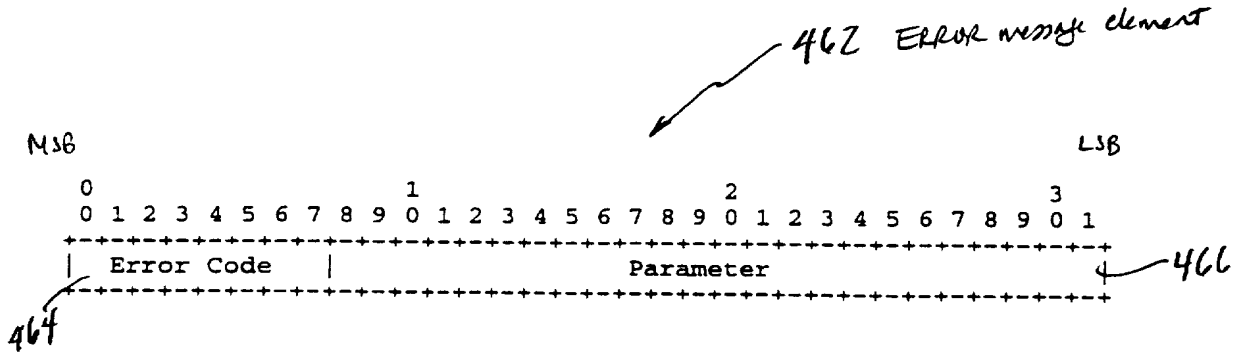

462 ERROR message element
MSB
LSB
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Error Code
Parameter
464
466

FIG. 9g

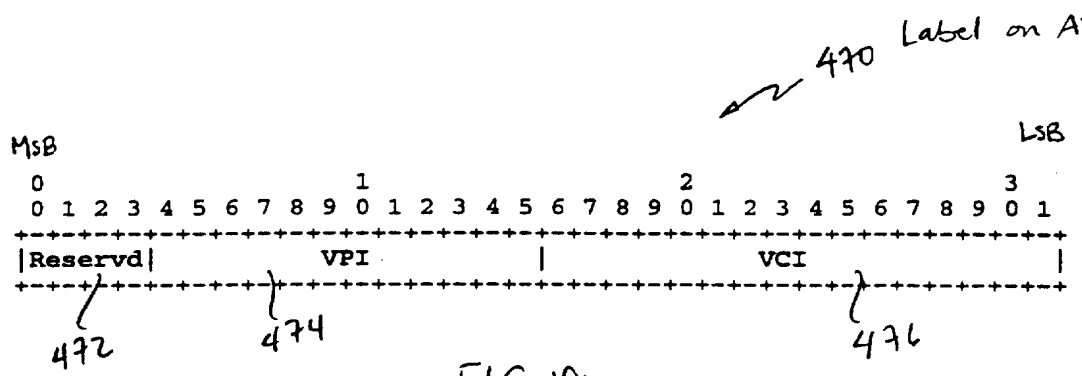


FIG. 10a

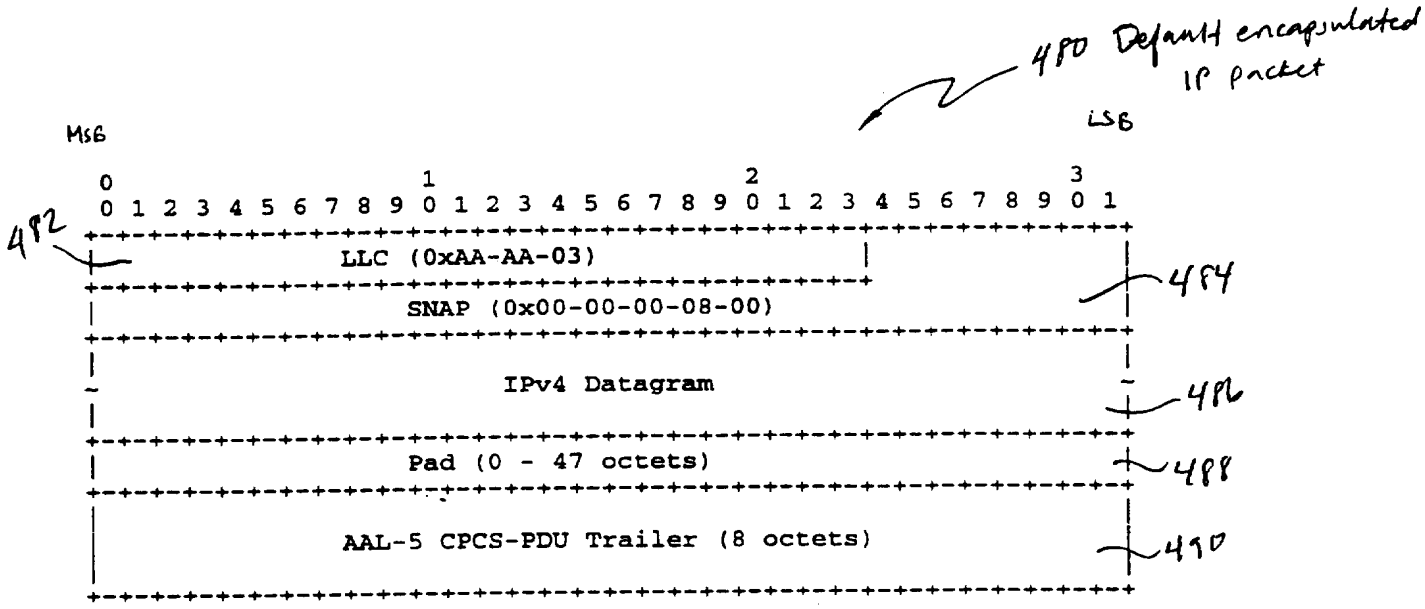


FIG. 10b

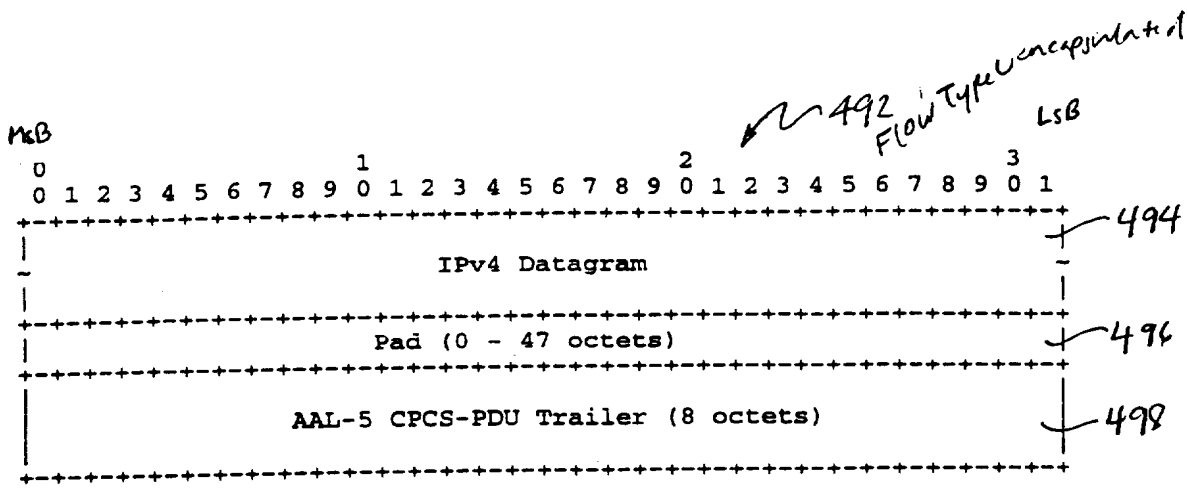


FIG. 10c

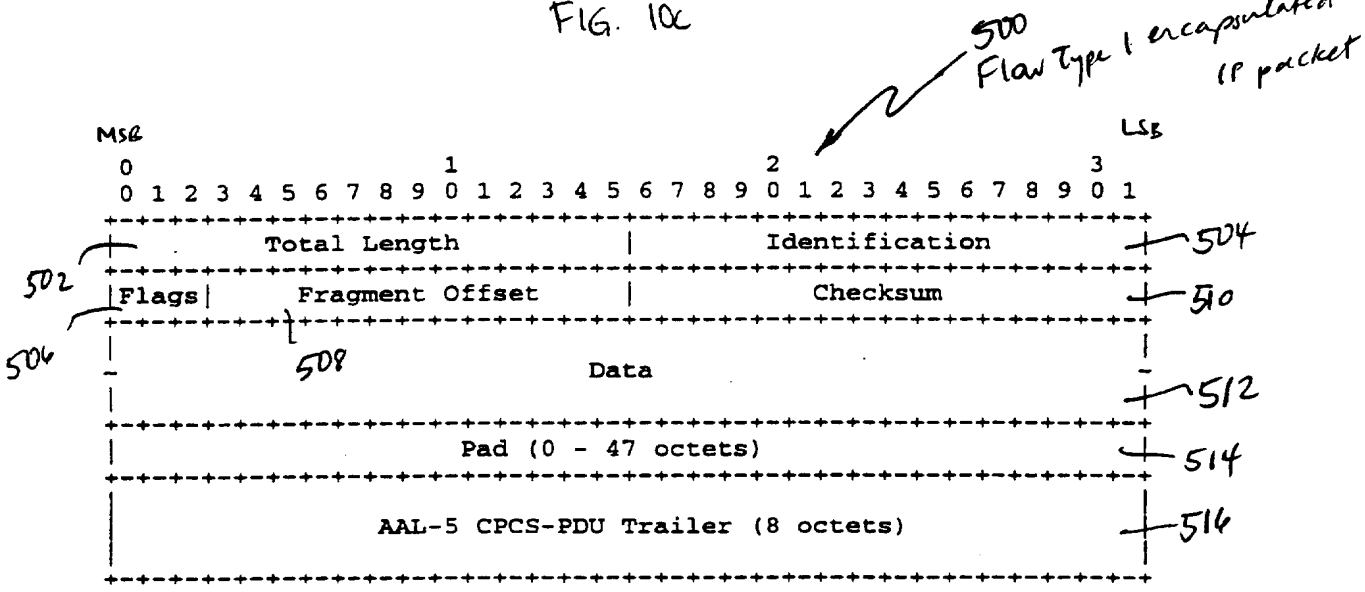


FIG. 10d

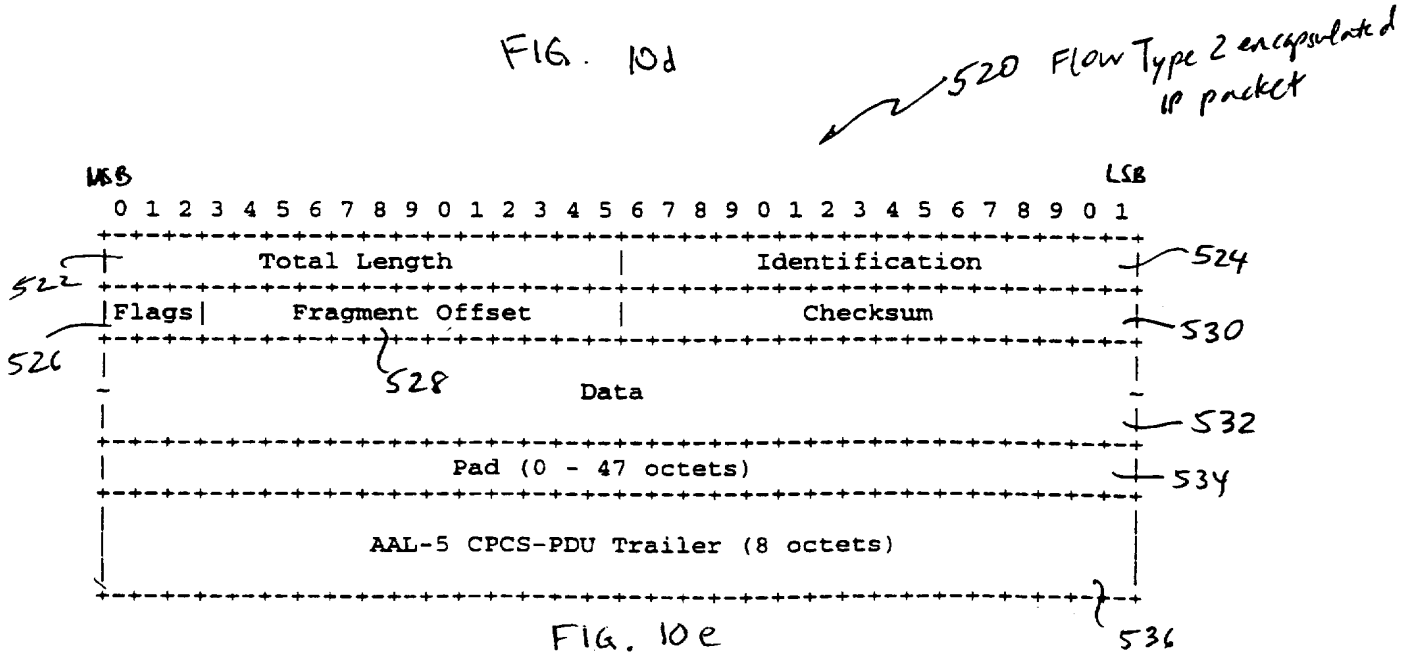


FIG. 10e

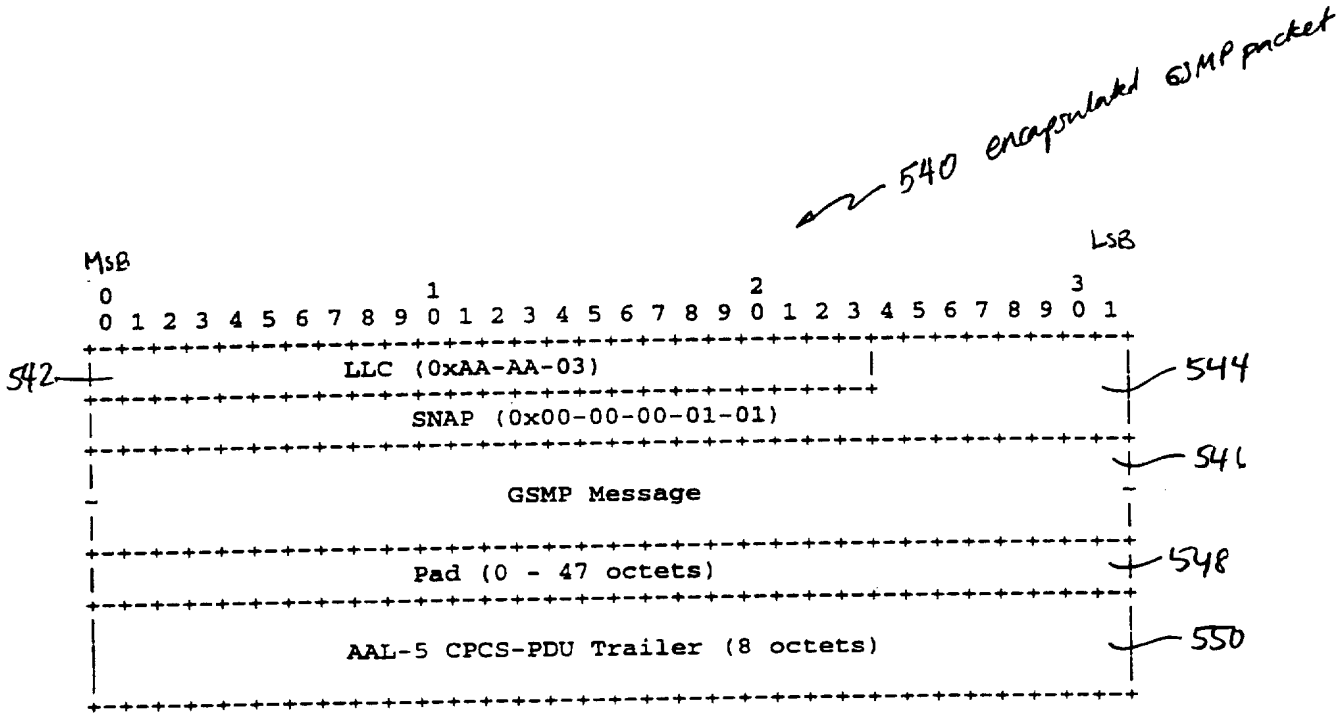

540 encapsulated GSMP packet
MSB
LSB
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
542
LLC (0xAA-AA-03)
544
SNAP (0x00-00-00-01-01)
546
GSMP Message
548
Pad (0 - 47 octets)
550
AAL-5 CPCS-PDU Trailer (8 octets)

FIG. 11a

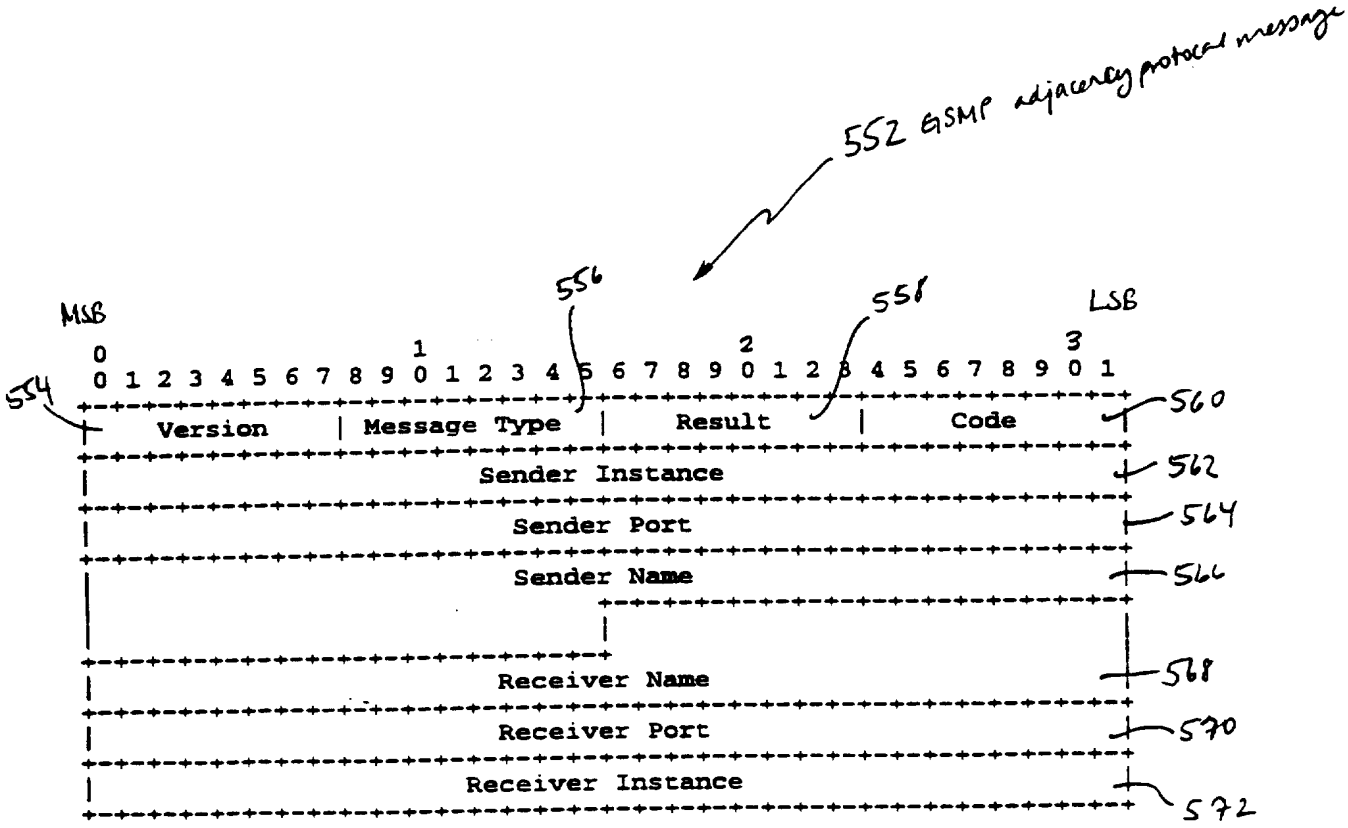

552 GSMP adjacency protocol message
556
558
MSB
LSB
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
554
Version
Message Type
Result
Code
560
Sender Instance
562
Sender Port
564
Sender Name
566
Receiver Name
568
Receiver Port
570
Receiver Instance
572

FIG. 11b

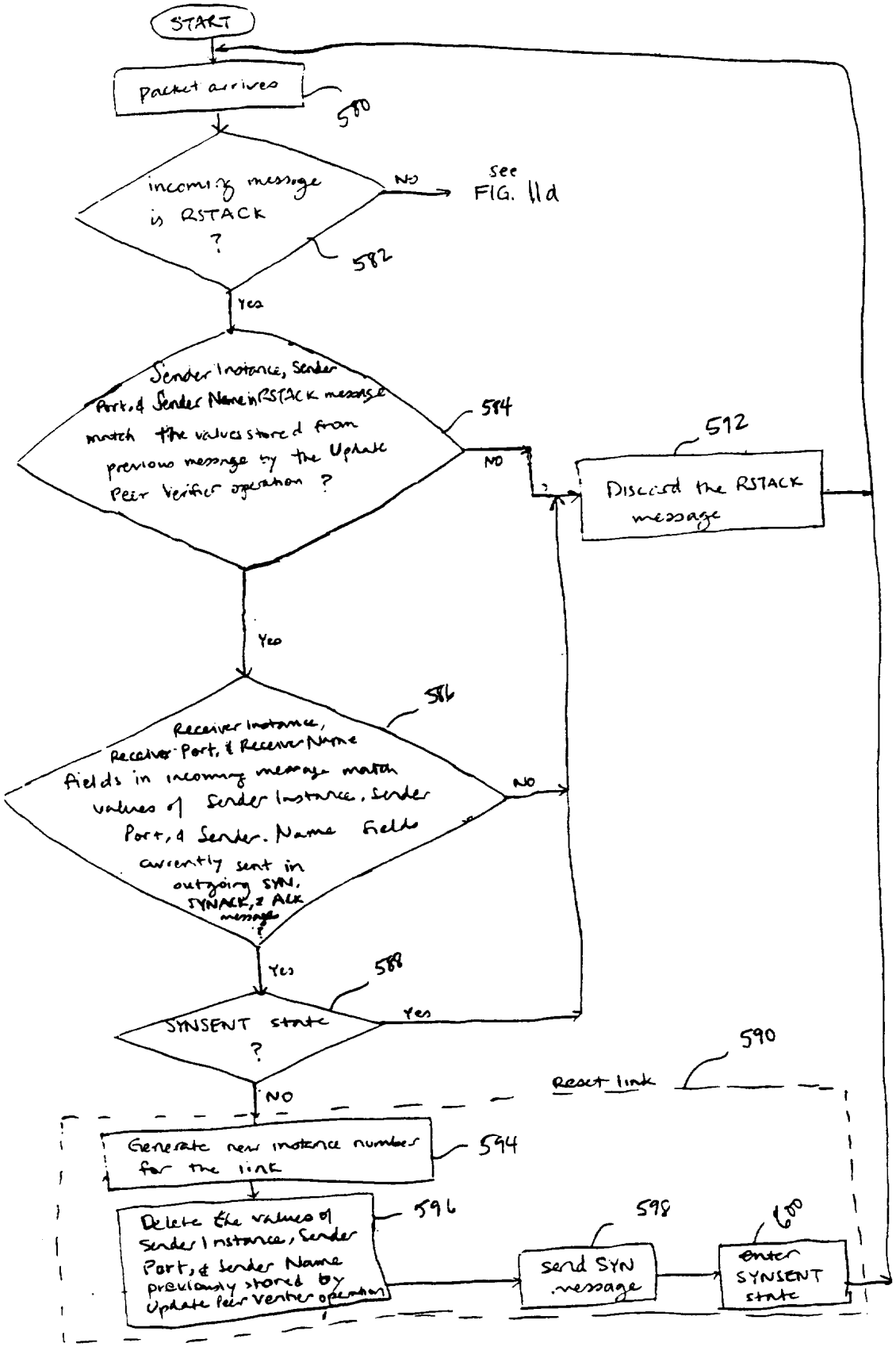
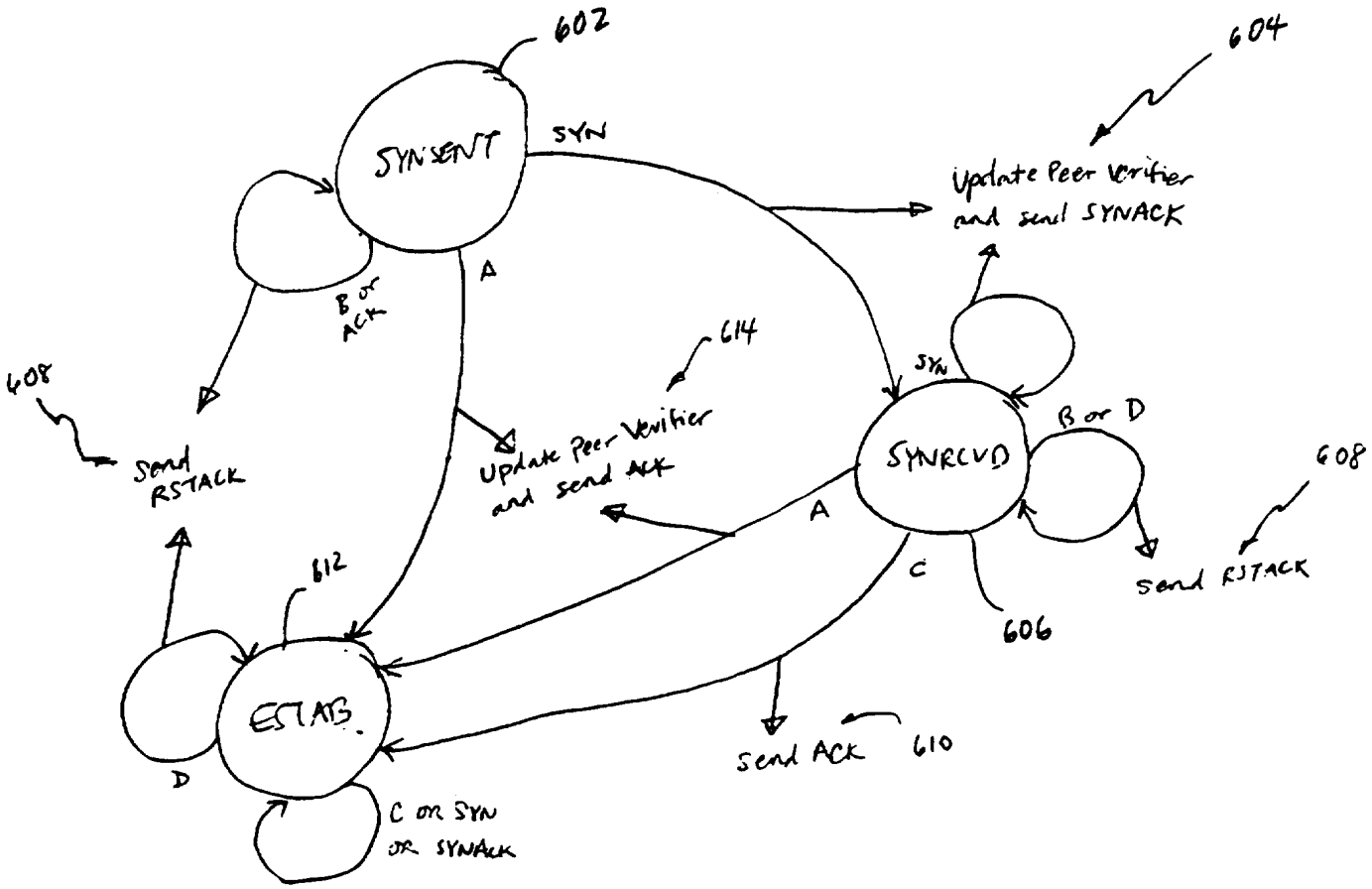

START
Packet arrives
580
incoming message is RSTACK ?
NO
see FIG. 11d
582
Yes
Sender Instance, Sender Port, & Sender Name in RSTACK message match the values stored from previous message by the Update Peer Verifier operation ?
584
NO
512
Discard the RSTACK message
Yes
Receiver Instance, Receiver Port, & Receiver Name fields in incoming message match values of Sender Instance, Sender Port, & Sender Name fields currently sent in outgoing SYN, SYNACK, & ACK messages ?
586
NO
Yes
SYNSENT state ?
588
Yes
NO
590
Reset link
Generate new instance number for the link
594
Delete the values of Sender Instance, Sender Port, & Sender Name previously stored by Update Peer Verifier operation
596
send SYN message
598
enter SYNSENT state
600

FIG 11c

Fig. 11a



Legend

A: SYNACK AND %C

B: SYNACK AND NOT(%C)

C: ACK AND %B AND %C

D: ACK AND NOT(%B AND %C)

where %B: Sender Instance, Sender Port, & Sender Name in incoming message match values stored from previous message by Update Peer Verifier operation

where %C: Receiver Instance, Receiver Port, & Receiver Name fields in incoming message match values of Sender Instance, Sender Port, & Sender Name currently sent in outgoing SYN, SYNACK, ACK messages.

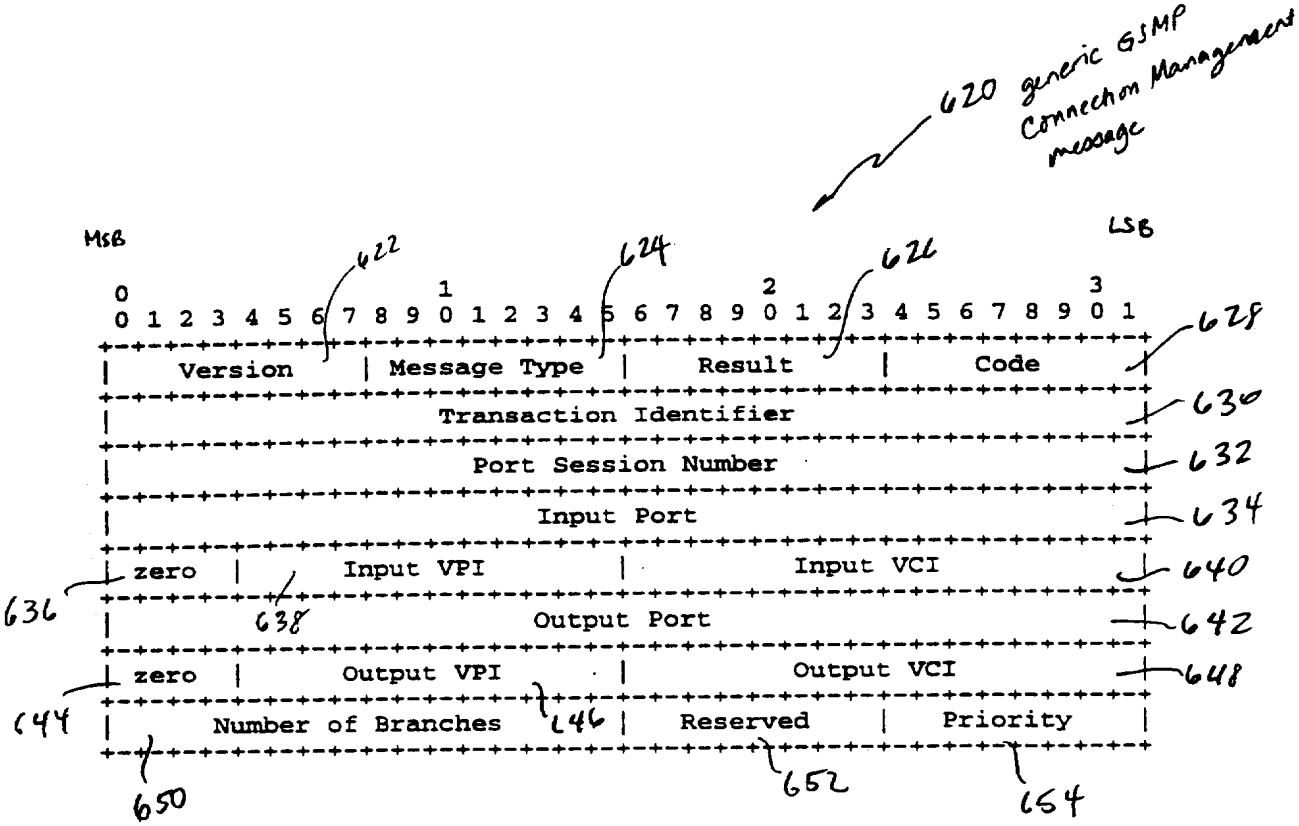

620 generic GSMP Connection Management message
MSB
LSB
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
622
624
626
628
Version
Message Type
Result
Code
630
Transaction Identifier
632
Port Session Number
634
Input Port
636
zero
Input VPI
Input VCI
640
638
Output Port
642
zero
Output VPI
Output VCI
648
644
Number of Branches
646
Reserved
Priority
650
652
654

FIG. 12

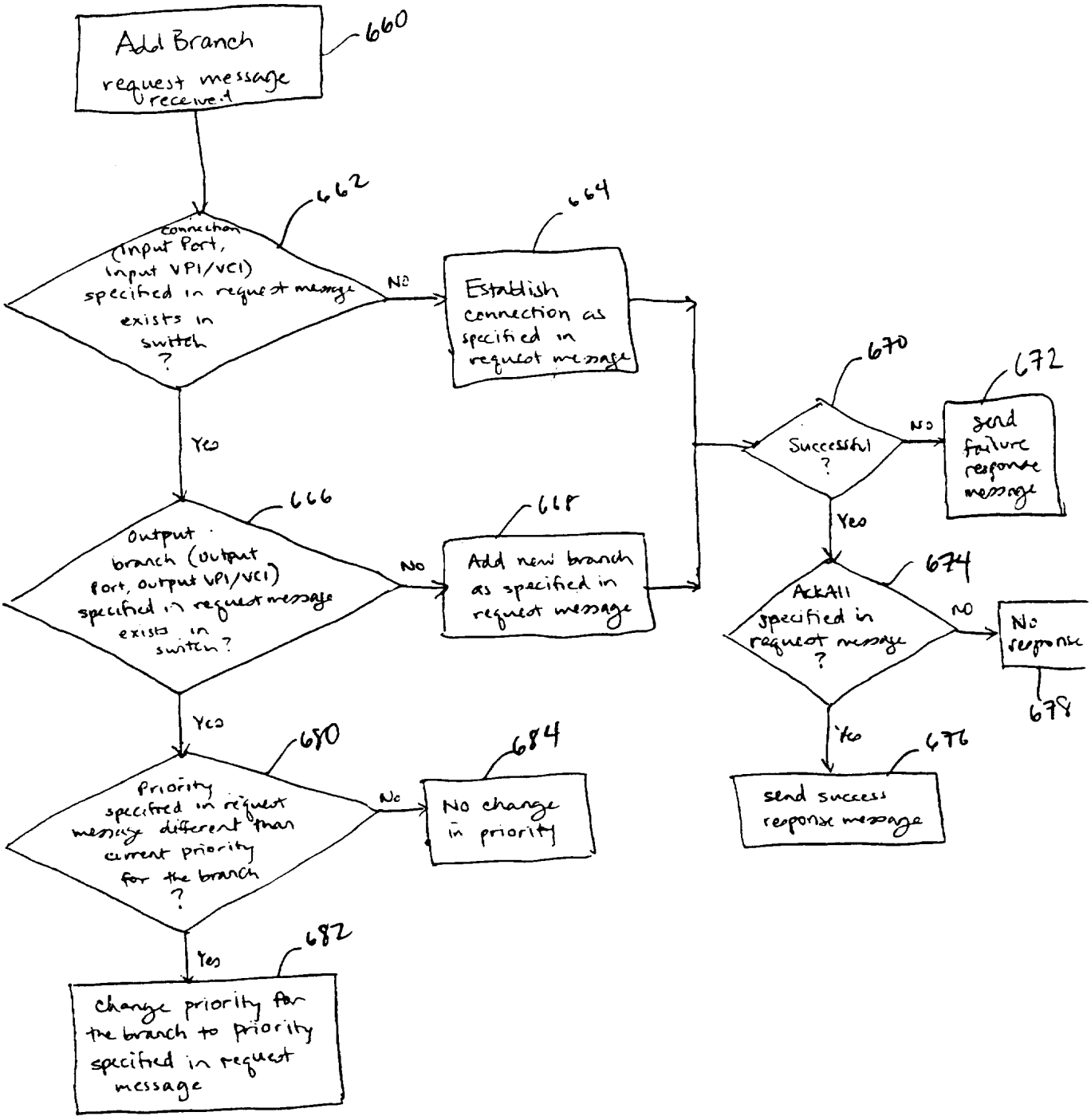

Add Branch request message received
660
Connection (Input Port, Input VPI/VCI) specified in request message exists in switch ?
662
No
Establish connection as specified in request message
664
Yes
Output branch (output port, output VPI/VCI) specified in request message exists in switch ?
666
No
Add new branch as specified in request message
668
Yes
Priority specified in request message different than current priority for the branch ?
680
No
No change in priority
684
Yes
change priority for the branch to priority specified in request message
682
Successful ?
670
No
send failure response message
672
Yes
AckAll specified in request message ?
674
NO
No response
678
Yes
send success response message
676

FIG. 13a.

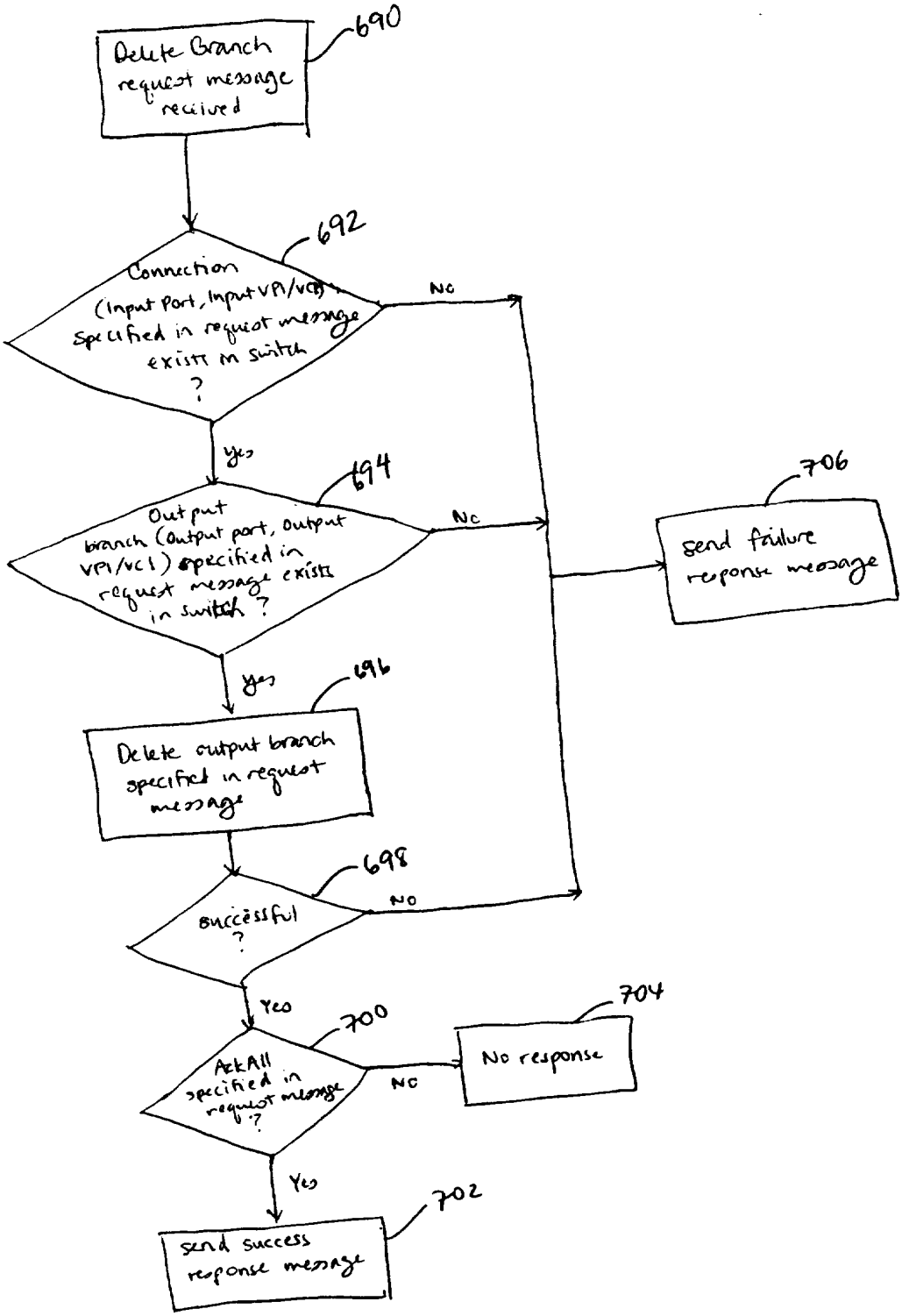

Delete Branch request message received
690
Connection (Input Port, Input VPI/VCI) specified in request message exists in switch ?
692
NO
Yes
Output branch (output port, output VPI/VCI) specified in request message exists in switch ?
694
NO
Yes
Send failure response message
706
Delete output branch specified in request message
696
successful ?
698
NO
Yes
AckAll specified in request message ?
700
NO
Yes
No response
704
send success response message
702

FIG. 13b

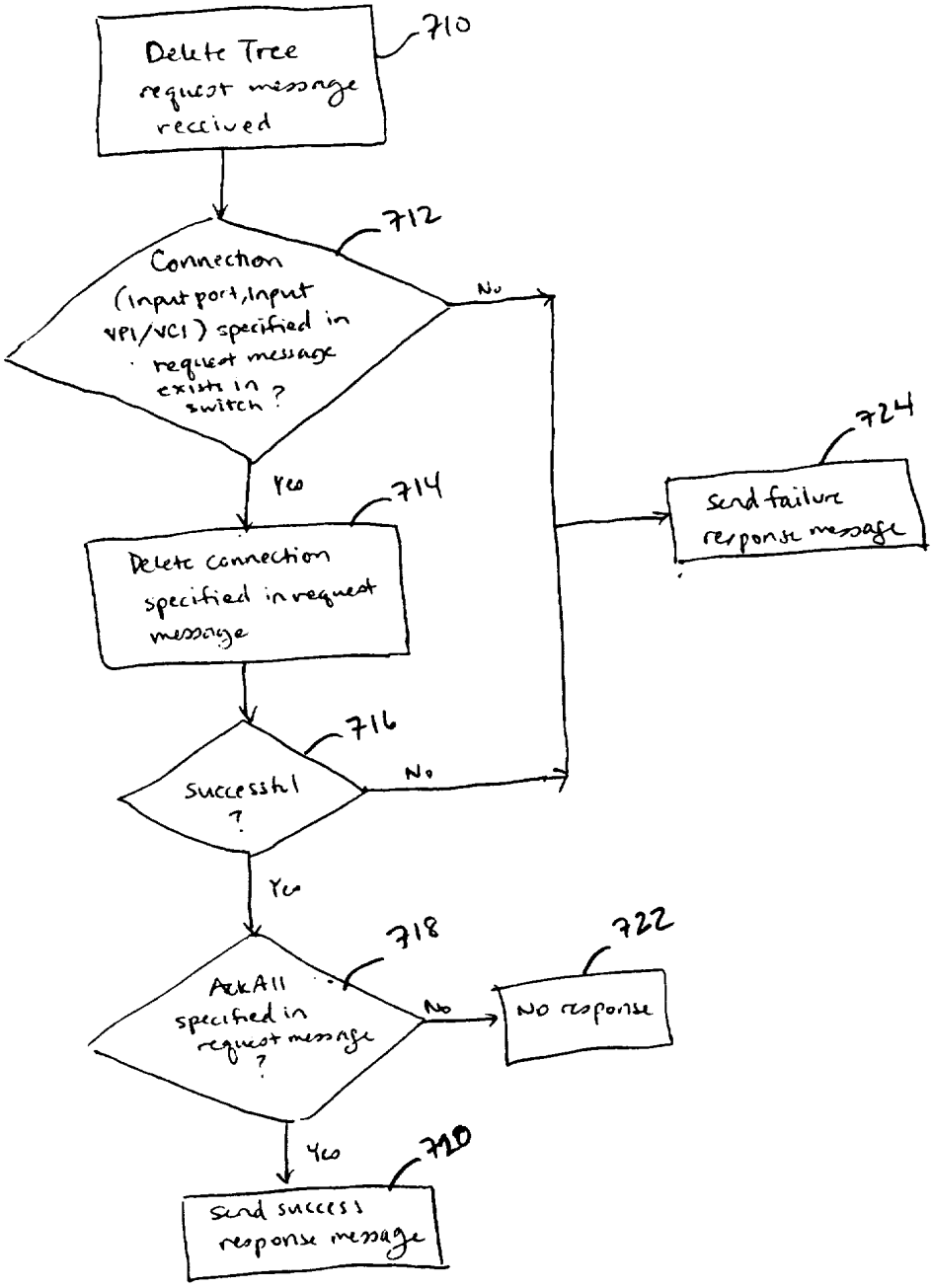

Delete Tree request message received
710
Connection (input port, input VPI/VCI) specified in request message exists in switch ?
712
No
Yes
Delete connection specified in request message
714
Send failure response message
724
Successful ?
716
No
Yes
AckAll specified in request message ?
718
No
No response
722
Yes
Send success response message
720

FIG. 13c

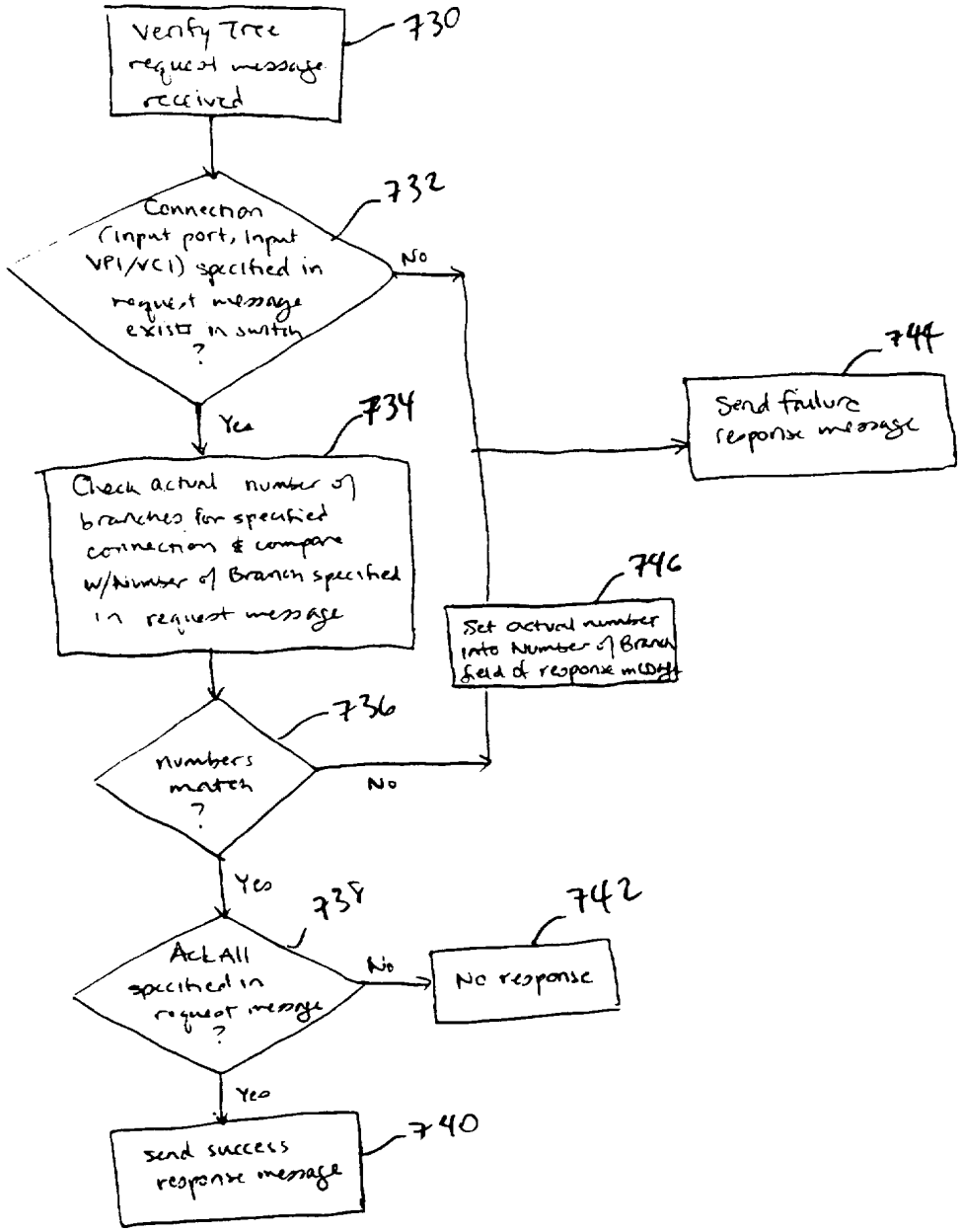

Verify Tree request message received
730
Connection (input port, input VPI/VCI) specified in request message exists in switch ?
732
No
Yes
744
Send failure response message
Check actual number of branches for specified connection & compare w/Number of Branch specified in request message
734
Set actual number into Number of Branch field of response message
746
Numbers match ?
736
No
Yes
AckAll specified in request message ?
738
No
No response
742
Yes
Send success response message
740

Fig. 13d

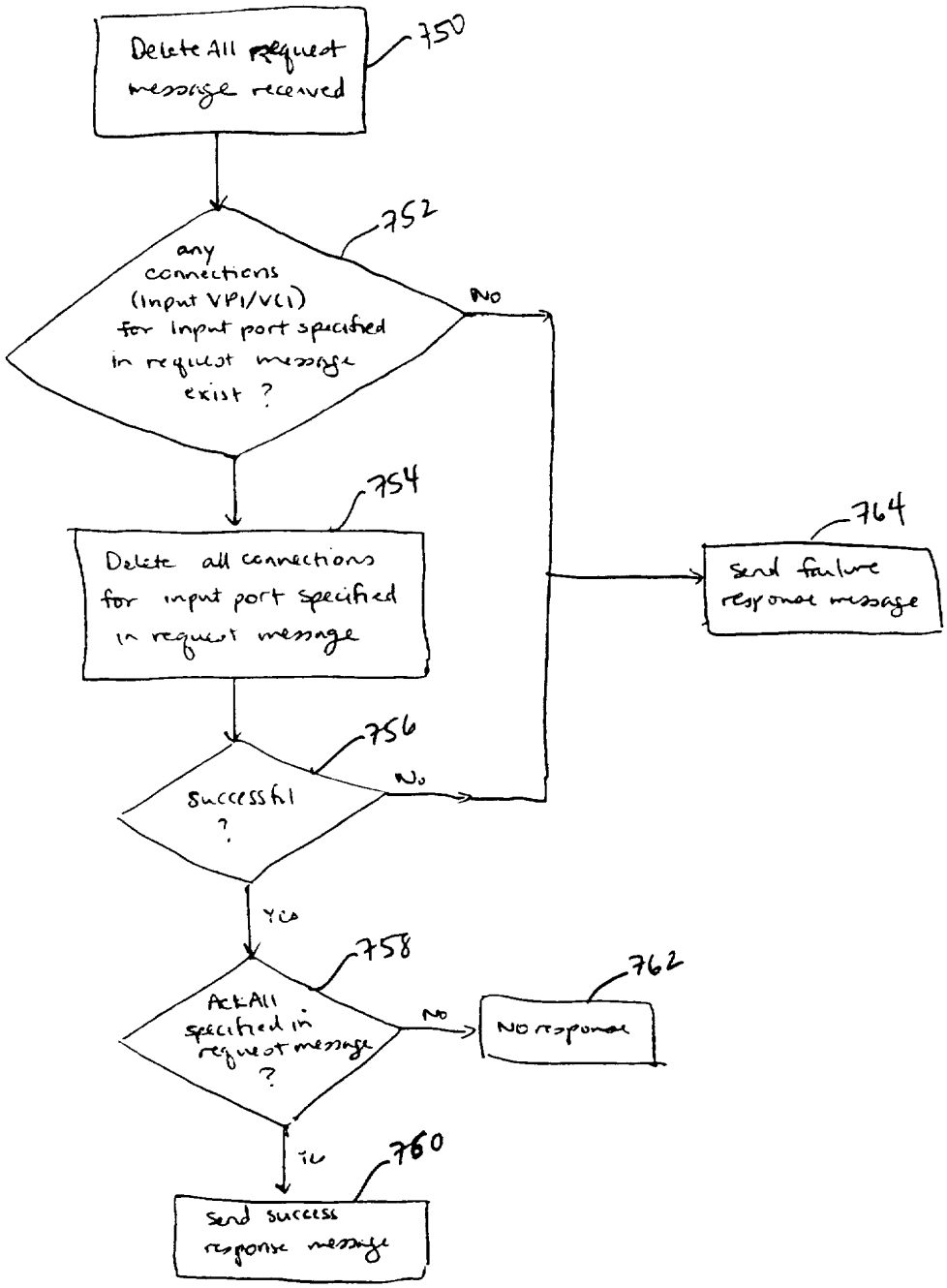

Delete All request message received
750
any connections (input VPI/VCI) for input port specified in request message exist ?
752
No
Delete all connections for input port specified in request message
754
Send failure response message
764
Successful ?
756
No
Yes
ACK All specified in request message ?
758
No
NO response
762
Yes
Send success response message
760

FIG. 13e

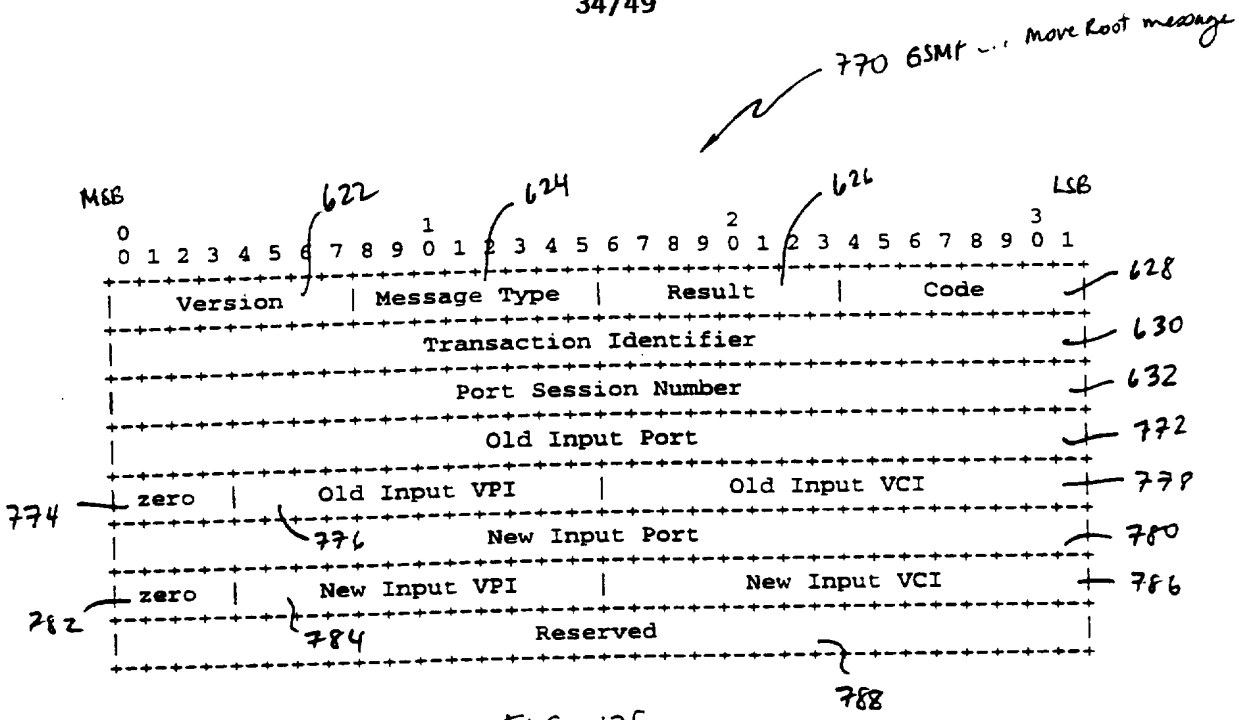


FIG. 13f

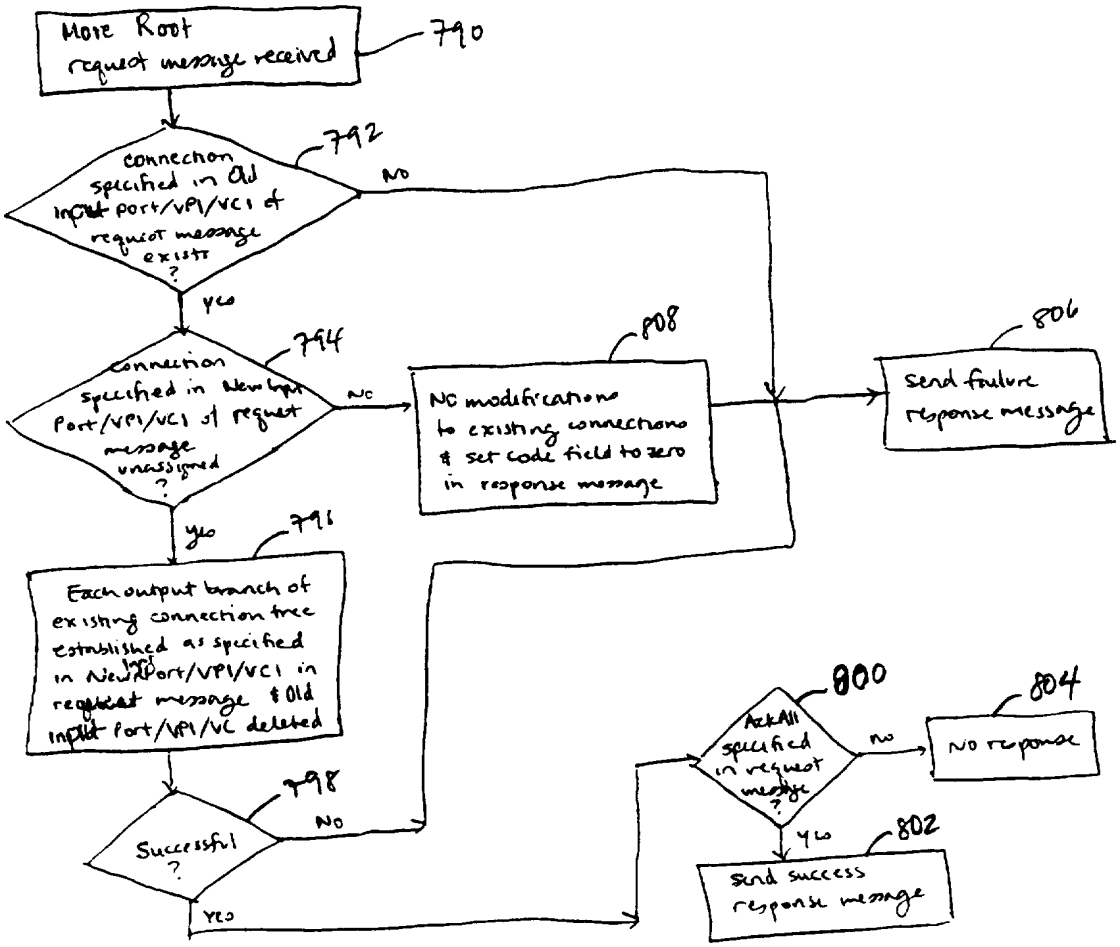


FIG. 13g

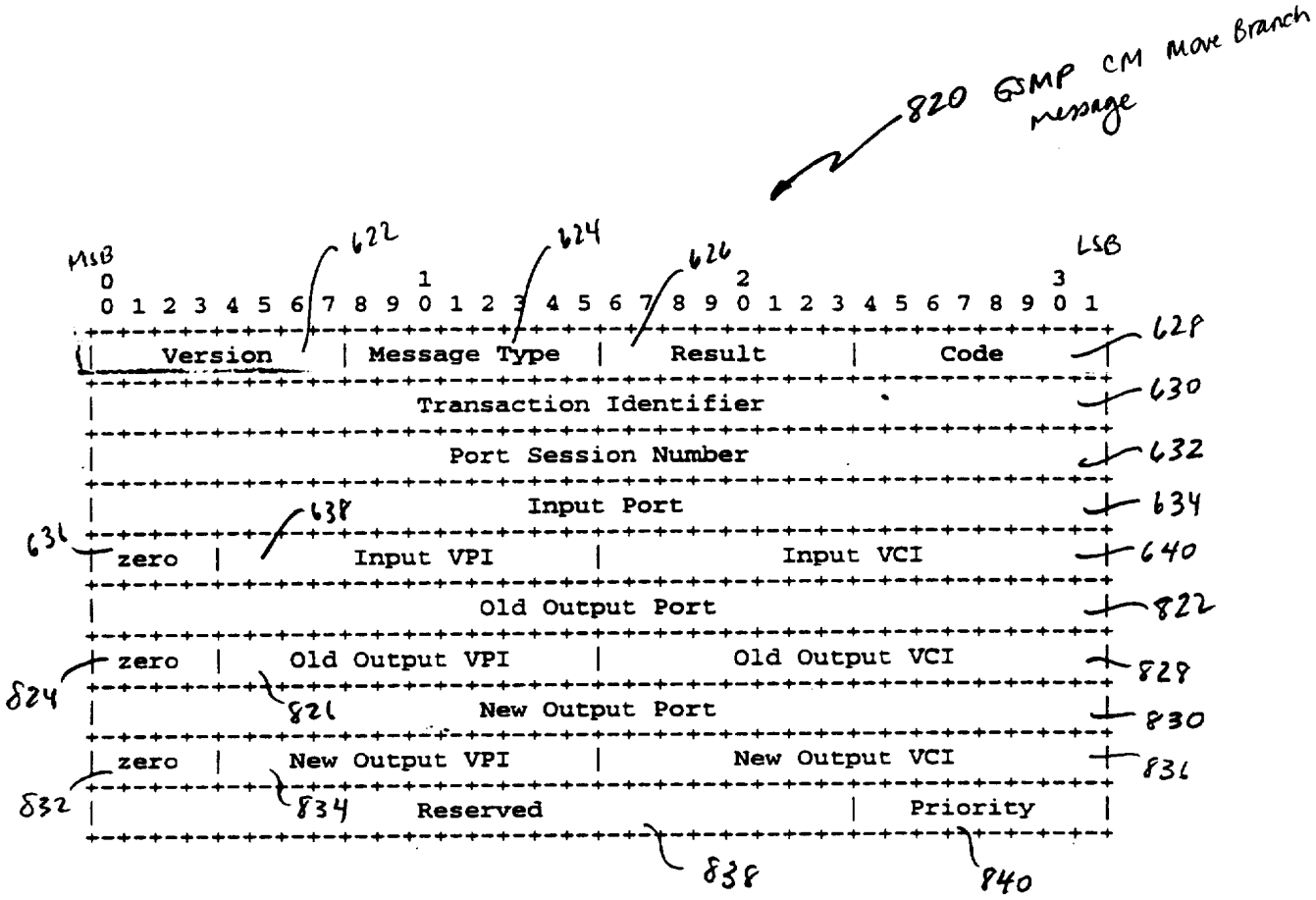

820 GSMP CM Move Branch message
MSB
LSB
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
622
624
626
Version
Message Type
Result
Code
628
Transaction Identifier
630
Port Session Number
632
638
Input Port
634
631
zero
Input VPI
Input VCI
640
Old Output Port
822
zero
Old Output VPI
Old Output VCI
824
828
821
New Output Port
830
zero
New Output VPI
New Output VCI
832
836
834
Reserved
Priority
838
840

FIG. 13h

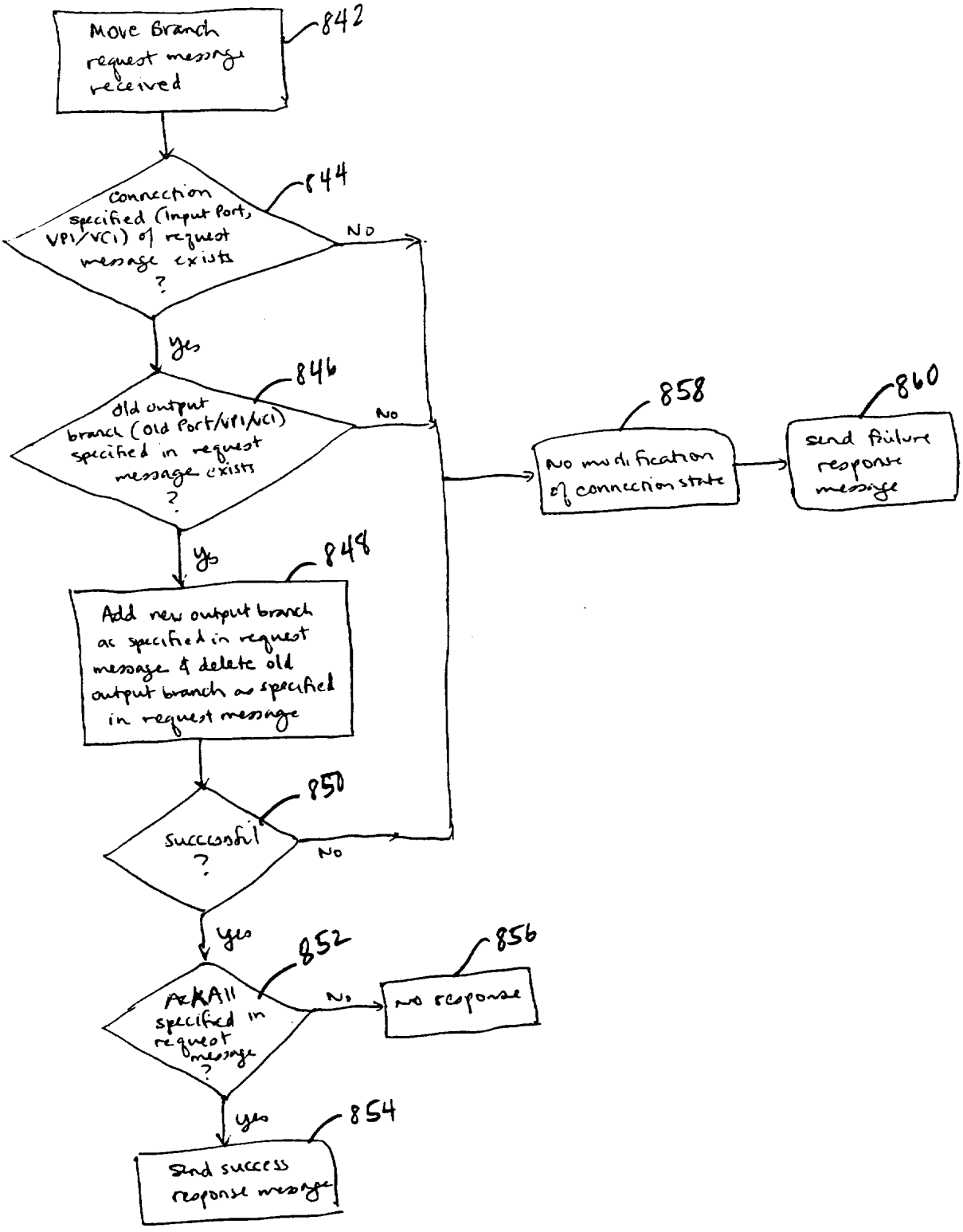

Move Branch request message received
842
Connection specified (Input Port, VPI/VCI) of request message exists ?
844
No
Yes
Old output branch (Old Port/VPI/VCI) specified in request message exists ?
846
No
Yes
No modification of connection state
858
send failure response message
860
Add new output branch as specified in request message & delete old output branch as specified in request message
848
Successful ?
850
No
Yes
AckAll specified in request message ?
852
No
no response
856
Yes
Send success response message
854

FIG. 13i

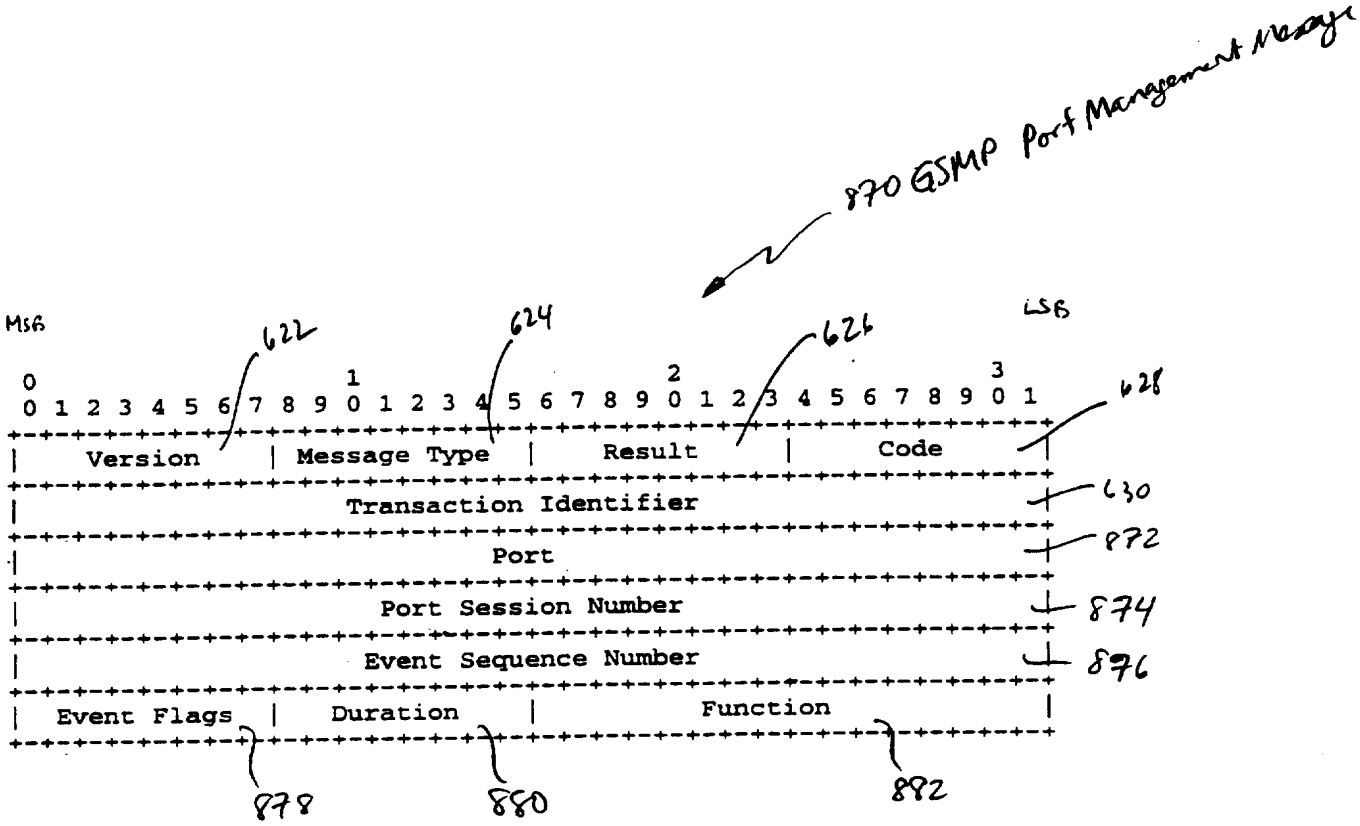

870 GSMP Port Management Message
MSB
LSB
622
624
626
628
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Version
Message Type
Result
Code
Transaction Identifier
630
Port
872
Port Session Number
874
Event Sequence Number
876
Event Flags
Duration
Function
878
880
882

FIG. 14

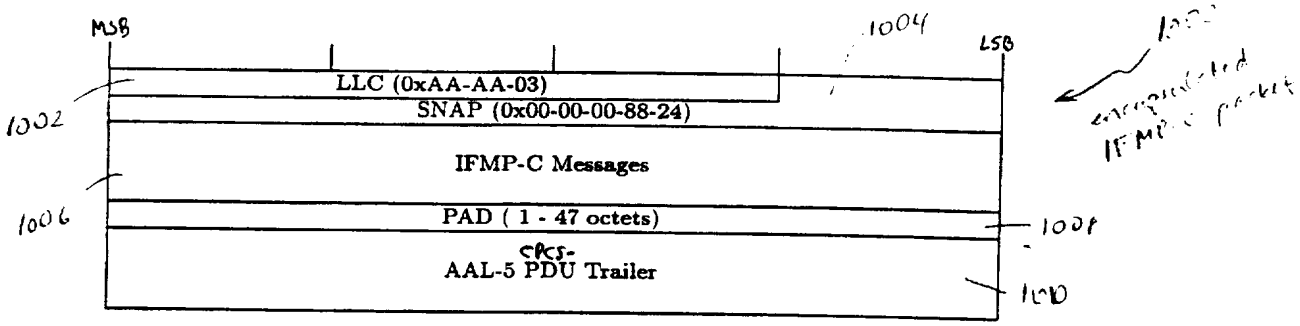


Fig. 15a

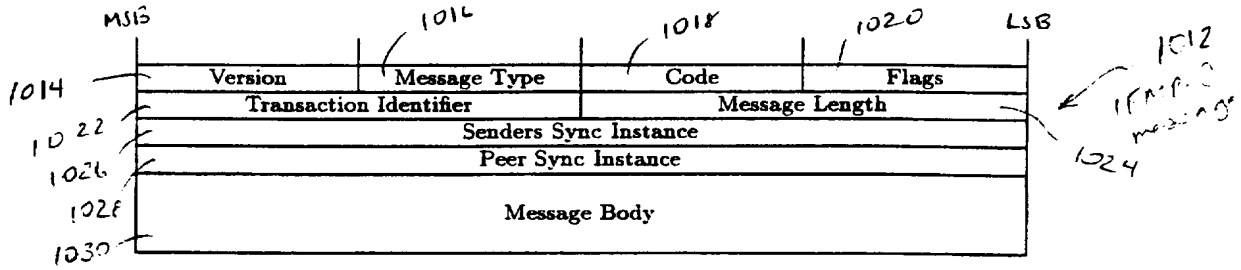


Fig. 15b

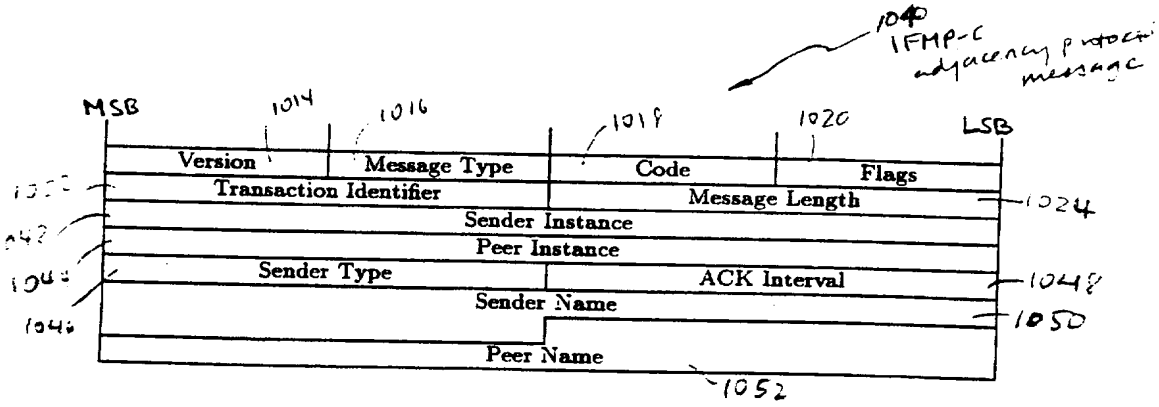
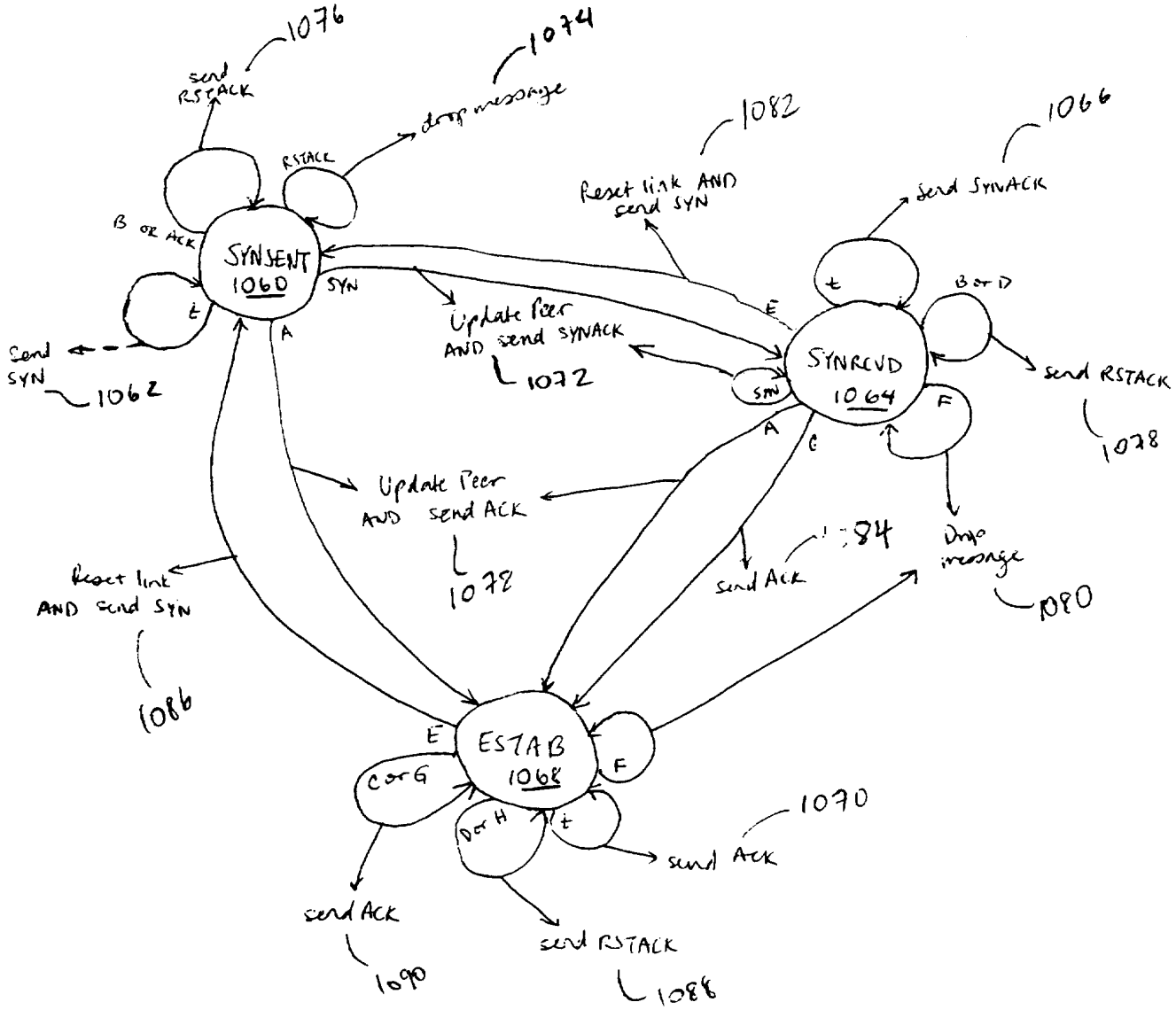


Fig. 16a



A: SYNACK AND ?.X

B: SYNACK AND NOT (?.X)

C: ACK AND ?.X AND ?.Y

D: ACK AND NOT (?.X AND ?.Y)

E: RSTACK AND ?.X AND ?.Y

F: RSTACK AND NOT (?.X AND ?.Y)

G: (SYN OR SYNACK) AND ?.X

H: (SYN OR SYNACK) AND NOT (?.X)

where ?.X = Peer Instance and Peer Name in received message match local values of instance and name associated with the link

?.Y = Sender Instance and Sender Name in received message match the values stored for the peer instance and peer name

FIG. 16b

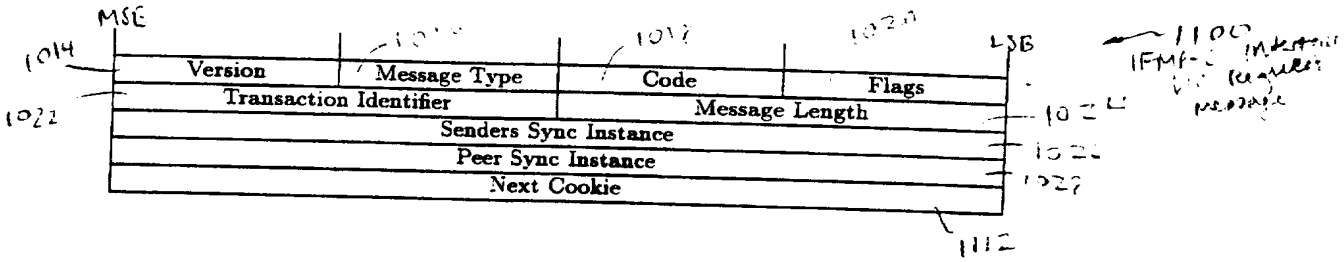


Fig. 17a

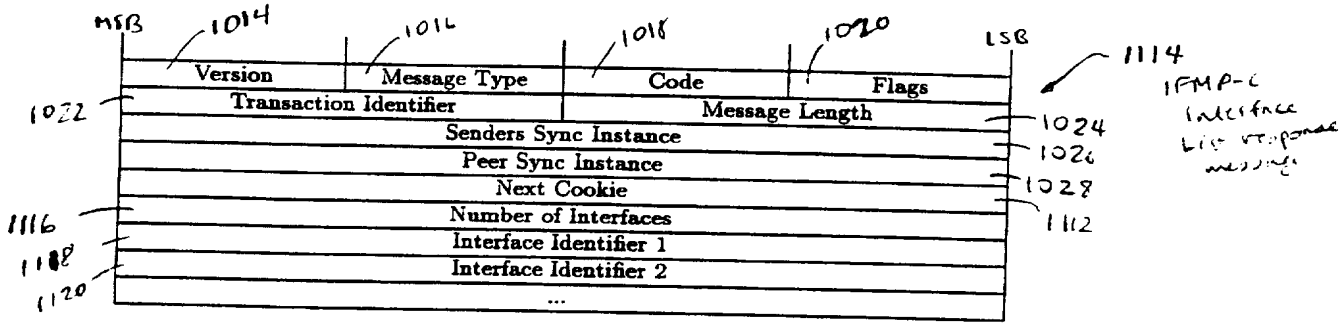


Fig. 17b

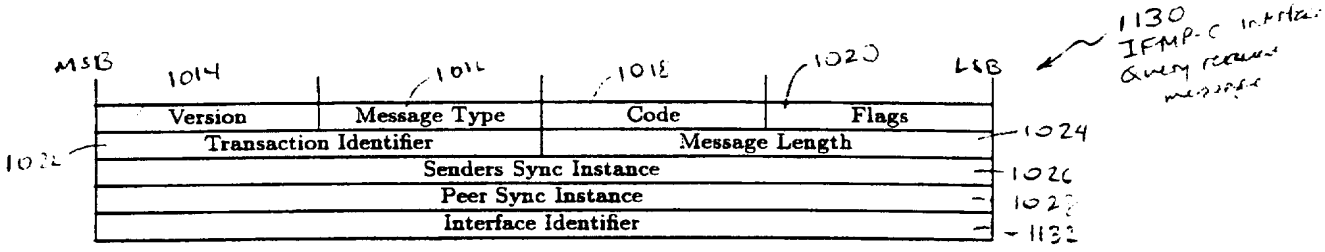


Fig. 17c

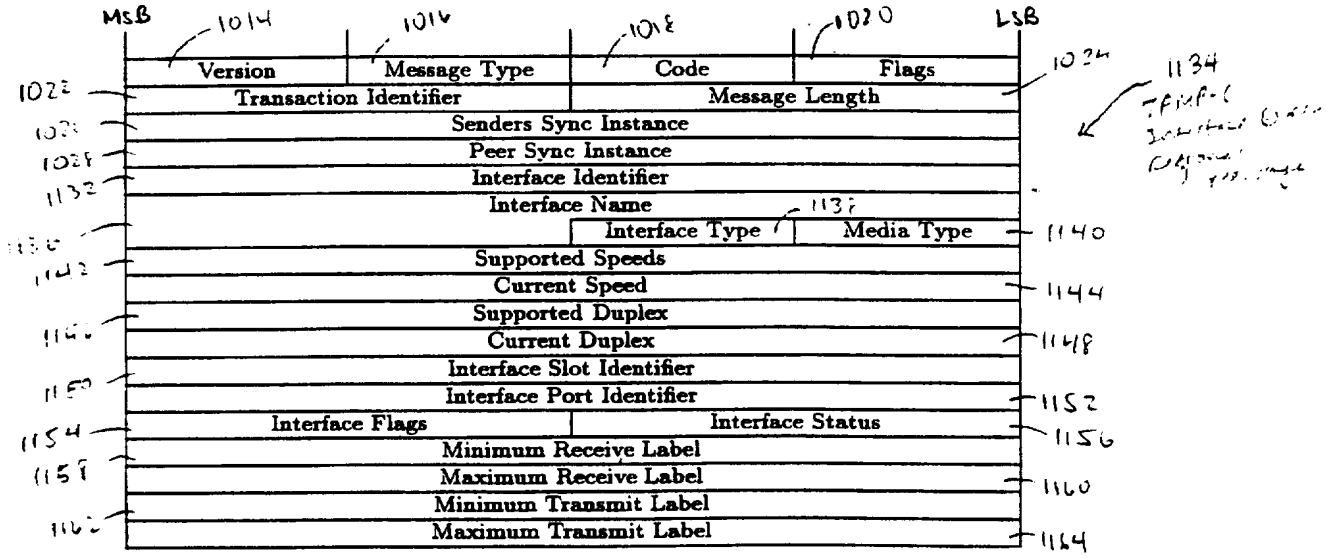


Fig. 17d

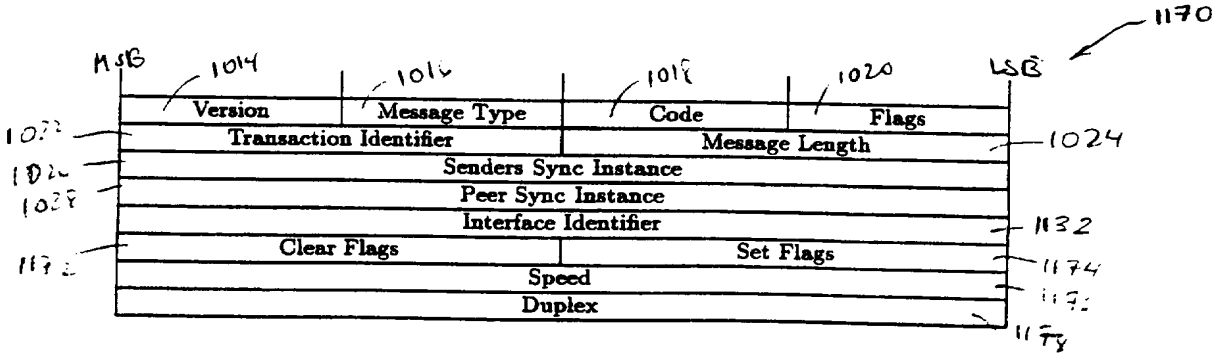


Fig. 17e

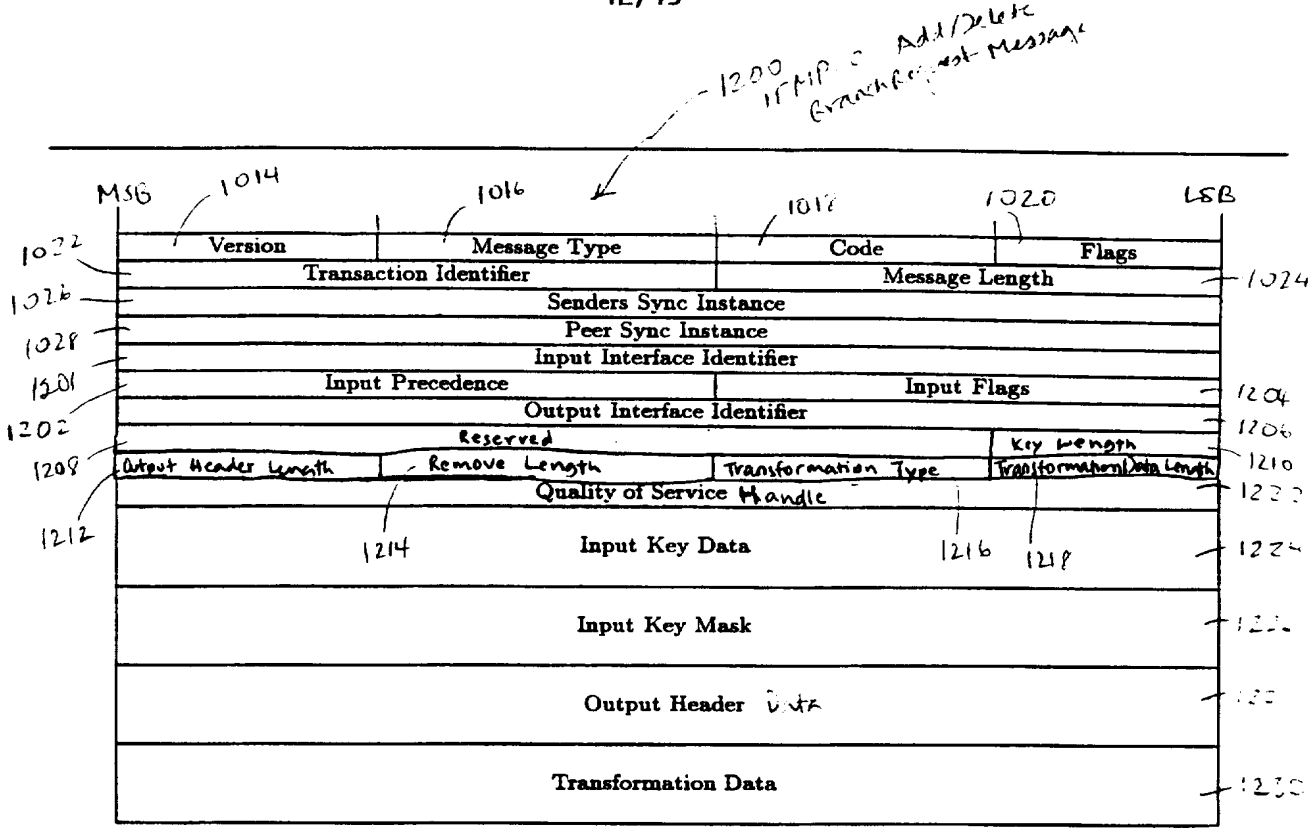


Fig. 18a

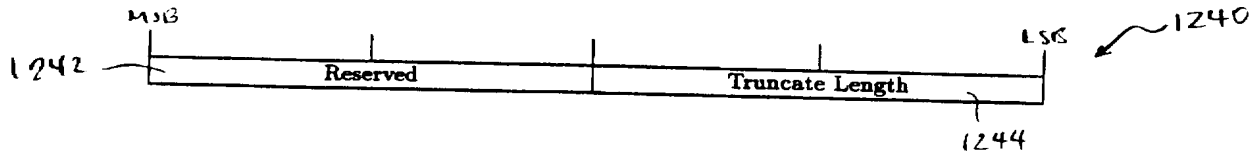


Fig. 18b

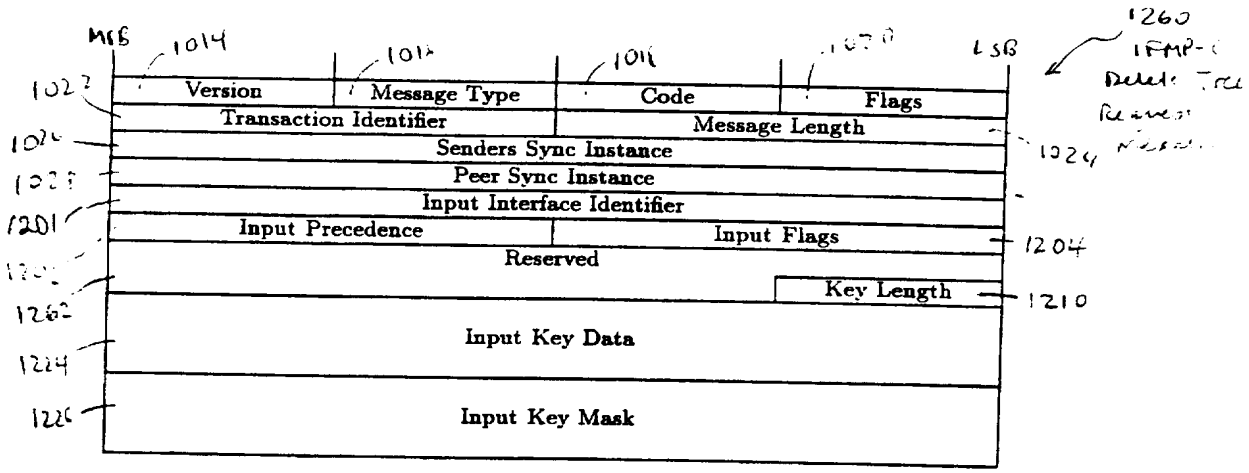


Fig. 18d

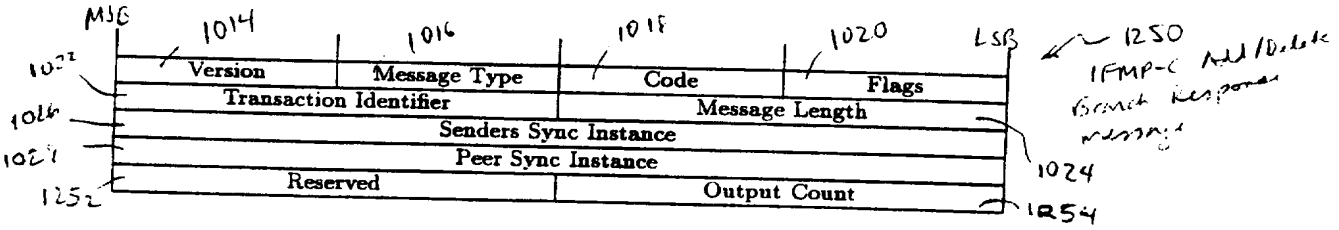


Fig. 18c

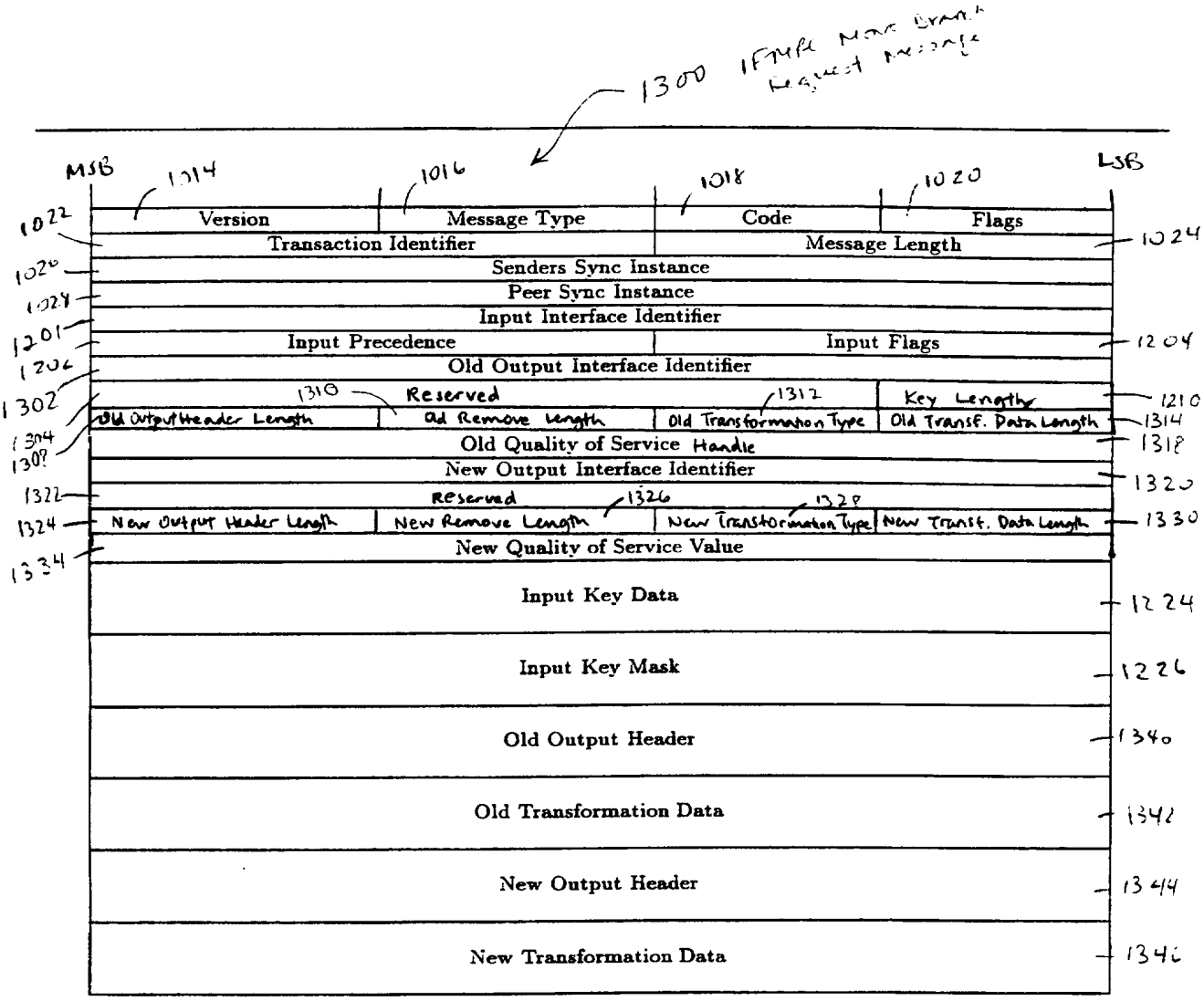

1300
MSB
LSB
1014
1016
1018
1020
Version
Message Type
Code
Flags
1022
Transaction Identifier
Message Length
1024
1026
Senders Sync Instance
1028
Peer Sync Instance
1201
Input Interface Identifier
Input Precedence
Input Flags
1204
1206
Old Output Interface Identifier
1302
1310
Reserved
1312
Key Length
1210
Old Output Header Length
Old Remove Length
Old Transformation Type
Old Transf. Data Length
1314
1304
1308
Old Quality of Service Handle
1318
New Output Interface Identifier
1320
1322
Reserved
1326
1328
1324
New Output Header Length
New Remove Length
New Transformation Type
New Transf. Data Length
1330
1334
New Quality of Service Value
Input Key Data
1224
Input Key Mask
1226
Old Output Header
1340
Old Transformation Data
1342
New Output Header
1344
New Transformation Data
1346

Fig. 18e

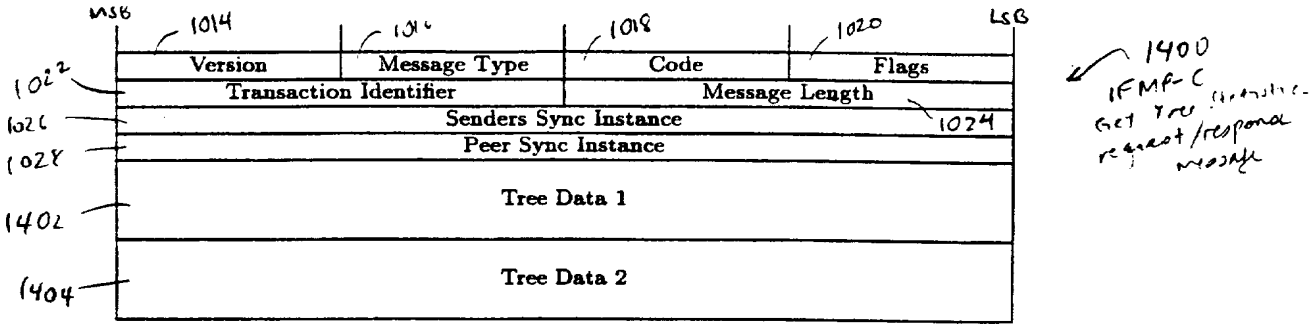


Fig. 19a

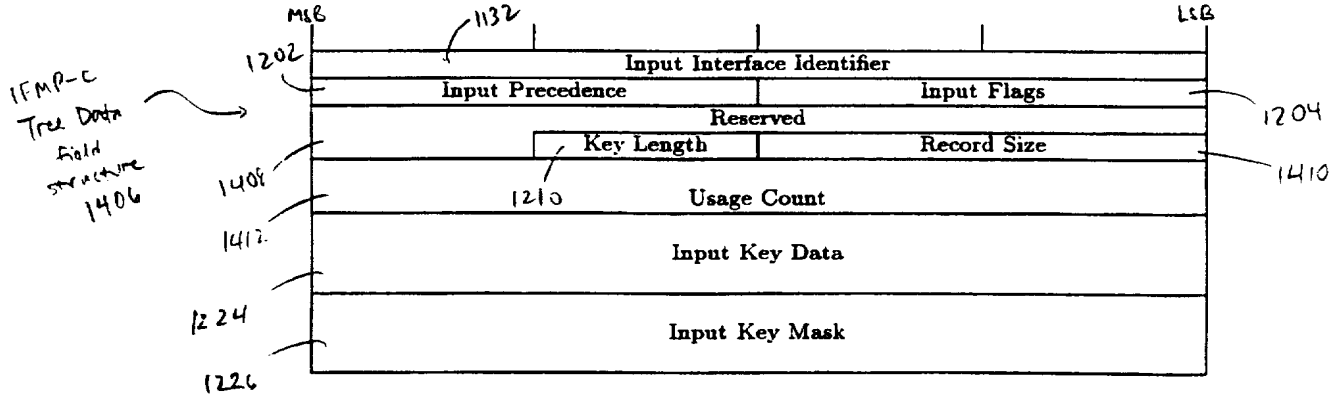


Fig. 19b

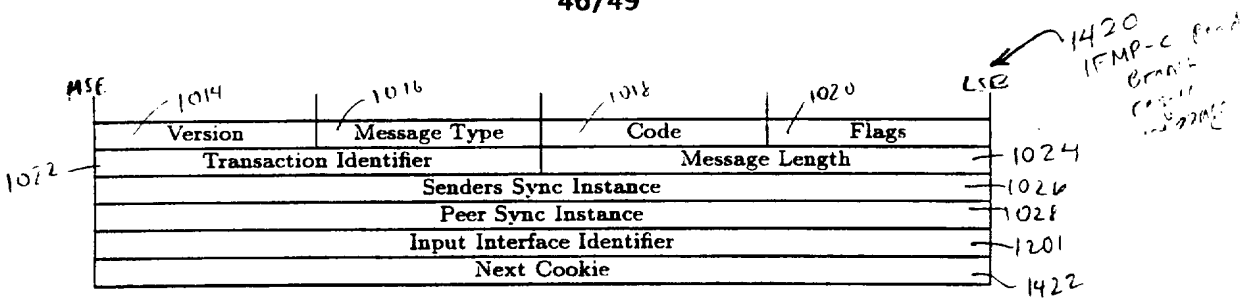


Fig. 20a

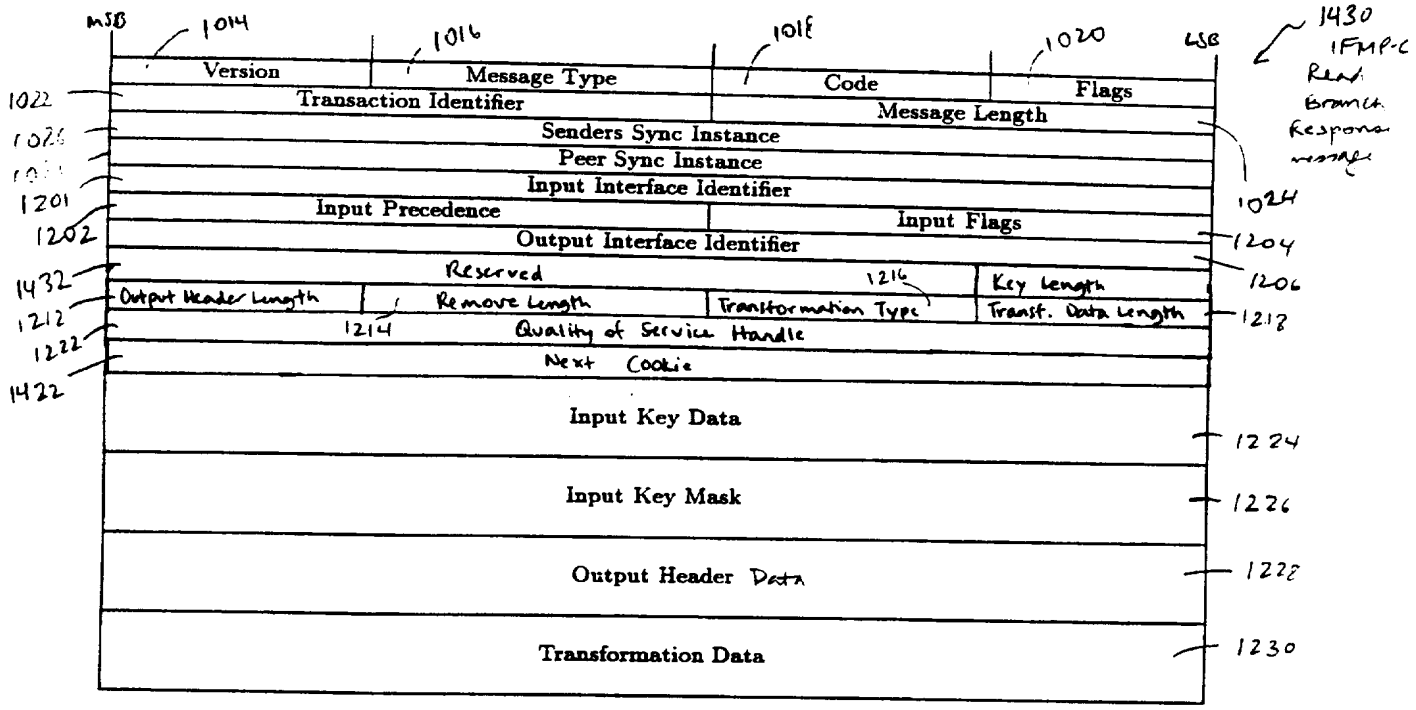


Fig. 20b

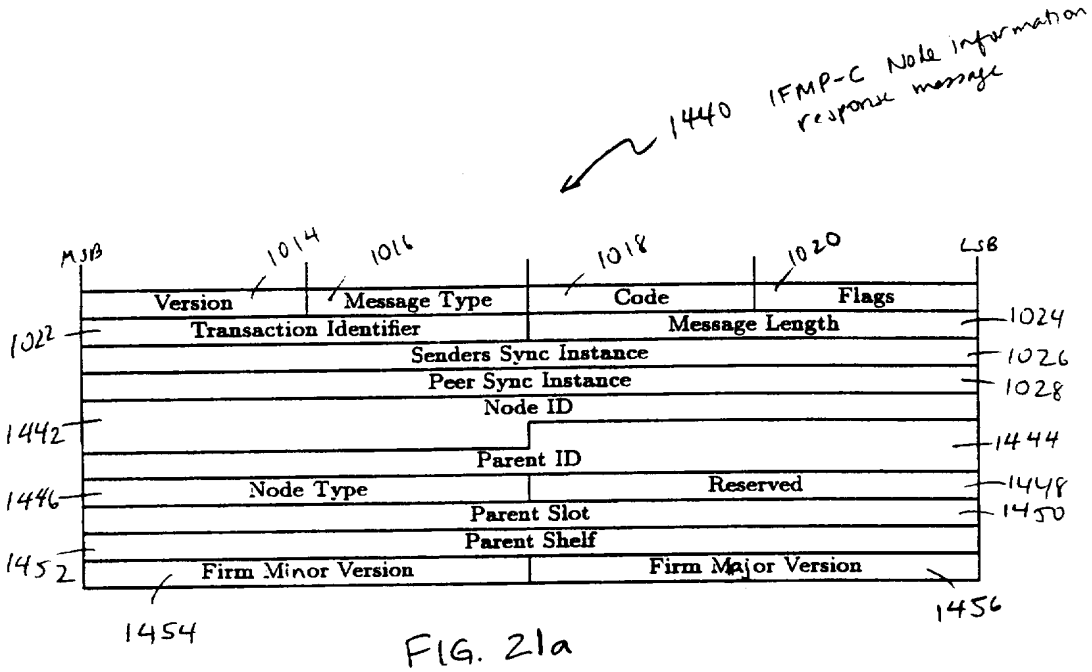


FIG. 21a

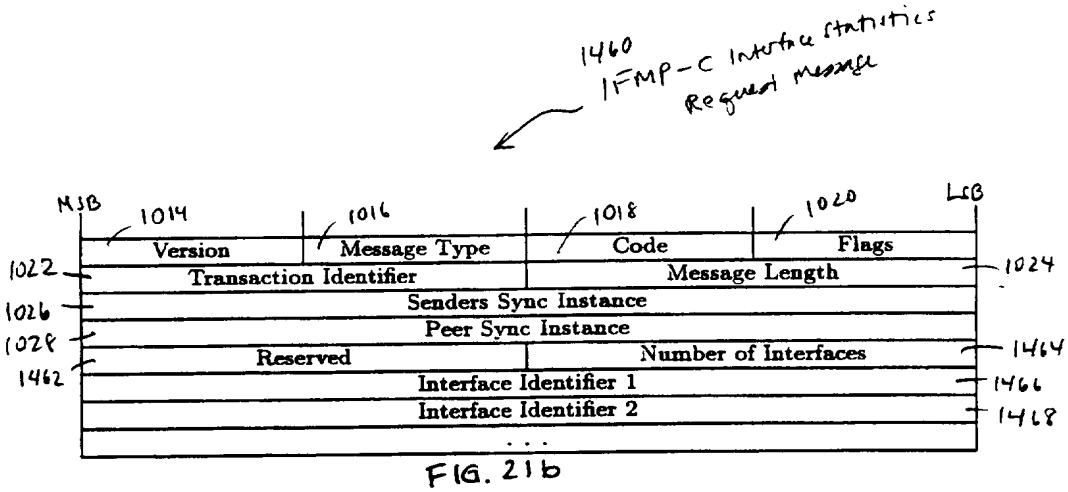


FIG. 21b

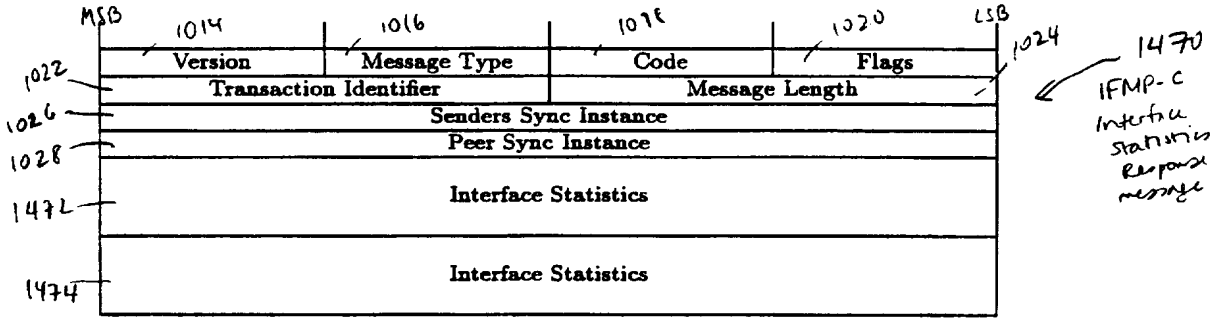


FIG. 21c

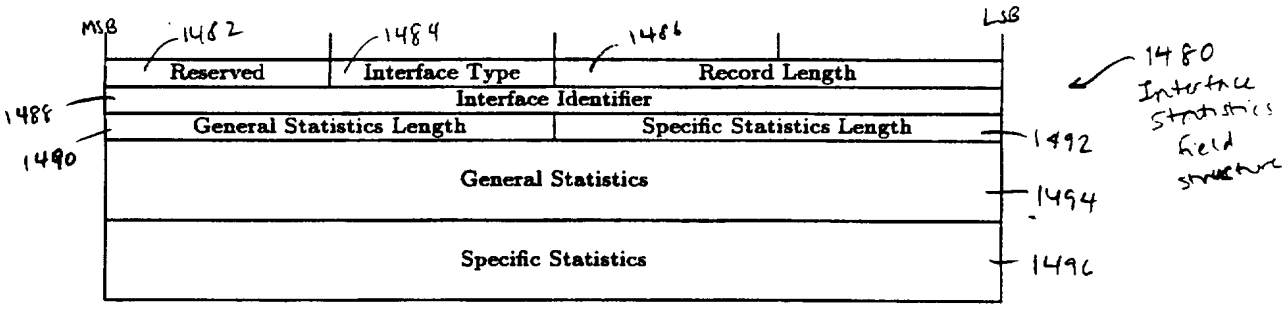


FIG. 21d

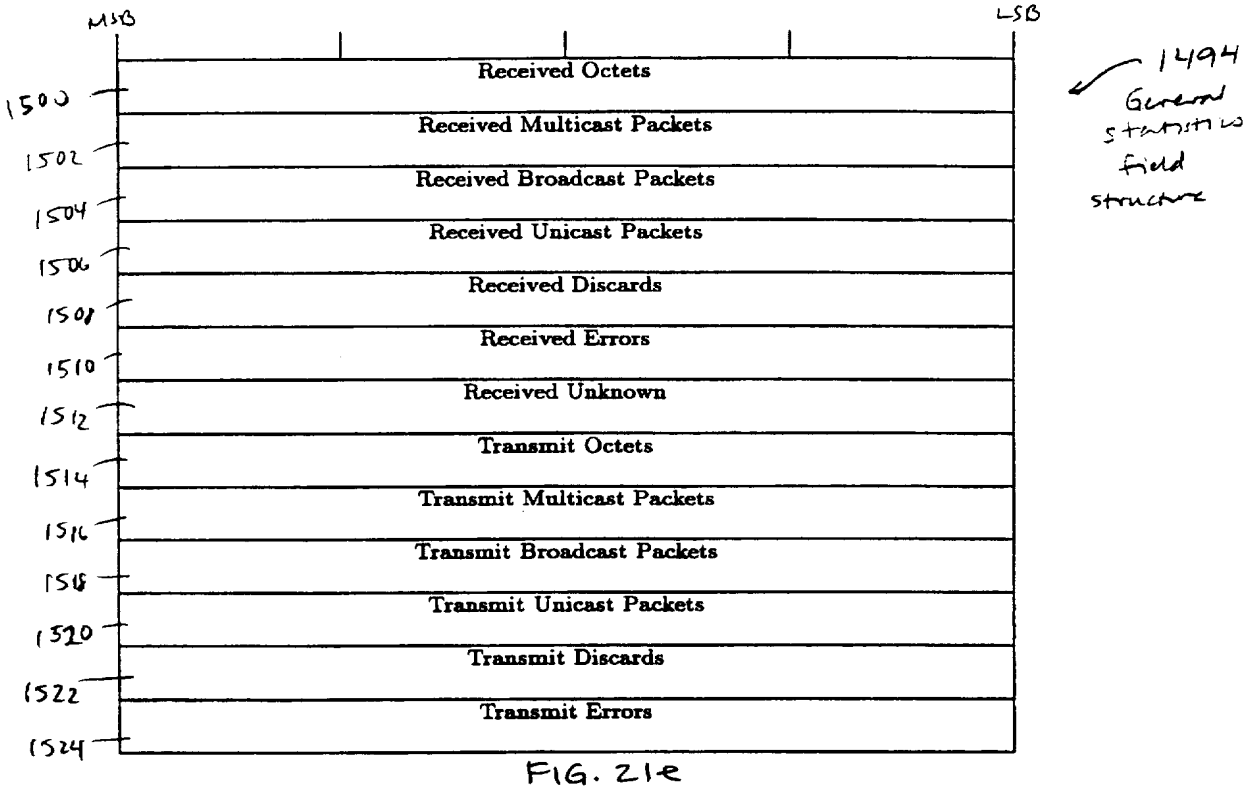


FIG. 21e

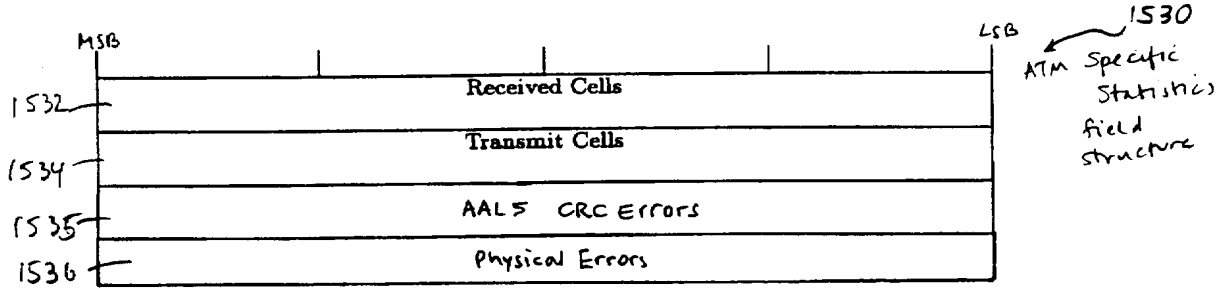


FIG. 21f

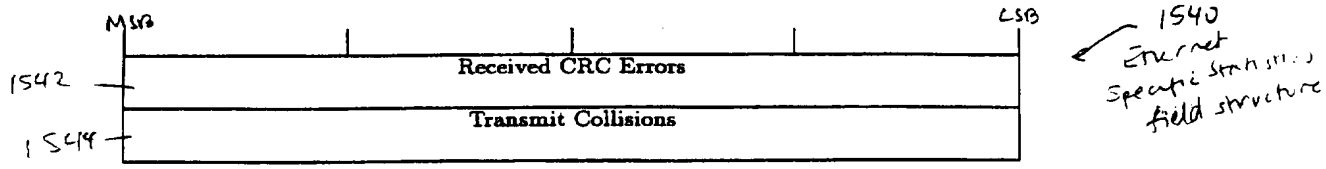


FIG. 21g

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/01595

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 13/14, 13/376; H04L 12/28, 12/407, 12/66, 12/56
US CL :395/200.15, 200.17, 200.16; 370/ 395, 396, 397, 399, 905
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/200.15, 200.17, 200.16; 370/ 395, 396, 397, 399, 905

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS, STN
Search terms: ATM, virtual circuit, IP protocol,

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, 5,444,702 A (BURNETT ET AL.) 22 August 1995; entire document	1-46
X --- Y	US, 5,379,297 A (GLOVER ET AL.) 03 January 1995 col. 6-17 figure 3-4	1-4, 8-11,13, 16-19, 23, 27-35, 38-45 --- 14-15,26
Y	SCOTT ET AL. 1990 Communications Support for Multimedia Workstations" pages 67-72 especially page 67	14-15
Y	JOHNSON 1995 "ATM Performance Management" pages 6/1-6/3	26

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:	
"A" document defining the general state of the art which is not considered to be of particular relevance	
"E" earlier document published on or after the international filing date	
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	
"O" document referring to an oral disclosure, use, exhibition or other means	
"P" document published prior to the international filing date but later than the priority date claimed	
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention	
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone	
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art	
"&" document member of the same patent family	

Date of the actual completion of the international search 11 APRIL 1997	Date of mailing of the international search report 0 2 JUN 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer LARRY DONAGHUE Telephone No. (703) 305-9500

Form PCT/ISA/210 (second sheet)(July 1992)*